MASTER

Domain Adaptation Graph Convolutional Network for Cross-Platform Cyberbully Detection

Zhao, Y.

Award date:
2020

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Mining Research Group

Domain Adaptation Graph Convolutional Network for Cross-Platform Cyberbully Detection

Master Thesis

Yuanzhi Zhao

Supervisors:
Prof. Dr. Mykola Pechenizkiy
Dr. Yulong Pei
External Assessor:
Dr. Robert Peharz

Eindhoven, October 2020

Acknowledgement

The completion of this master project is undoubtedly exciting and relieving. Its process is definitely educative and fruitful, but also arduous and brings intermittent stress, especially at the beginning stage. With the worsening Corona outbreak in the middle of its process, this project takes a considerably long period of time to finish, even longer than the usual one-year maximum duration. Thus, I must thank my two supervisors, Mykola Pechenizkiy and Yulong Pei, for their careful instructions from the most difficult, the very beginning stage. I have greatly increased both academic skills and experience in handling this kind of long and relatively large project. I will take these, as well as the self-confidence accumulated through the project with me, whatever the field I will delve into in the future.

Abstract

The occurrences of cyberbully phenomena has increased significantly due to the dramatic growth of social platforms and their users. The amount of available data for cyberbully detection research does not keep pace with the social platform development. Therefore, a cross-platform cyberbully detection technique using existing labeled data from one social platform to examine data from another platform for potential cyberbully instances will be very helpful. In this project, we apply an newly proposed domain adaptation method-DAGCN(**G**raph **C**onvolutional **N**etworks for **D**omain **A**daptation) to the field of cross-platform cyberbully detection. The experiments based on this application provides valuable data for evaluating the generalization of DAGCN to the cyberbully detection realm. The experiment topic also includes comparing graph construction method and alleviating the class imbalance issue for disproportionate data.

Contents

List of Figures **v**

List of Tables **vi**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Research Content . . . 2
1.3 Research Questions . . . 2

2 Related Work **4**
2.1 Cyberbully Detection . . . 4
2.2 Domain Adaptation . . . 5
2.3 Graph-based Semi-supervised Learning . . . 6

3 Problem Statement **7**

4 Framework **9**
4.1 Framework of DAGCN . . . 9
4.2 Framework of the Project . . . 10
4.2.1 Data Cleaning . . . 10
4.2.2 Sentence to Embedding . . . 10
4.2.3 Network Construction . . . 13
4.2.4 Semi-Supervised Learning . . . 14
4.2.5 Time Complexity Analysis . . . 16

5 Experiment Setup **17**
5.1 Goal of Experiment . . . 17
5.2 Dataset . . . 17
5.3 Data Cleaning&Embedding Conversion . . . 18
5.4 Baseline Methods . . . 18
5.5 Oversampling Schemes . . . 19
5.6 Implementation Details . . . 20
5.6.1 Network Construction . . . 20
5.6.2 Semi-supervised Learning . . . 20
5.7 Computer System&Hardware . . . 21

6 Result Analysis **22**
6.1 Preparation . . . 22
6.1.1 Introduction to Sign Test . . . 22
6.1.2 Introduction to Holm-Bonferroni Method . . . 23
6.1.3 Introduction to Experiment Variations and Result Metrics . . . 23
6.1.4 Processing Invalid Results . . . 25
6.1.5 Normality Test for All Data . . . 25

6.2 Comparison Between Two Graph Construction Methods 27
6.3 Comparison Between DAGCN and Baselines 28
6.3.1 Comparison Among Baseline Methods 29
6.3.2 Comparison Among DAGCN Variations 30
6.3.3 Comparison Between DAGCN and Baseline 30
6.4 Comparison Regarding Oversamplings 31
6.4.1 Comparison of Oversamplings and No Oversamplings 31
6.4.2 Comparison of Oversamplings Methods 34
6.5 Comparison Between Word Embedding Methods 34
6.6 Comparison About the Data Pairs' Origins 35
6.7 Individual Cases Analysis 36

7 Conclusion **40**
7.1 Main Contributions 40
7.2 Limitations 41
7.2.1 Definition of Cyberbully Detection 41
7.2.2 Dataset Selection 41
7.2.3 Graph Construction 42
7.2.4 Baseline 42
7.2.5 Statistic Test 42
7.3 Future Work 42

Bibliography **44**

Appendix **48**

A Full List of Oversampling Cases **48**

B Source-bridge edges and Bridge number For the First Construction Method **50**

C List of Cases Included for Comparison Between No Sampling, 1st Order Normal Oversampling and 2nd Order Normal Oversampling **53**

D List of Cases Included in ADASYN Oversampling and maGAN oversampling **54**

E Normality Test Results **55**

F All Experiment Results **57**

List of Figures

1.1 Illustration of Relationship Between Cross Platform and Domain Adaptation . . . 2

4.1 DAGCN Structure . . . 9
4.2 Framework Illustration . . . 10
4.3 Process of Embedding Conversion . . . 11
4.4 Illustration of CBOW and SkipGram model.The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. [30] . . . 11
4.5 Illustration of Word Similarity Assessment from article [36] . . . 12
4.6 Illustration of BERT's Input Representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. [18] . . . 13
4.7 Illustration of GraphSAGE's Sampling and Aggregation from [21] . . . 14
4.8 Illustration of FastGCN's Integral Estimation from [9] . . . 15

5.1 ADDA Framework . . . 19
5.2 Structure of Classifier and Encoder . . . 19
5.3 FastGCN Structure with Discriminator . . . 21

6.1 Auxiliary Illustration of the Concept of Precision and Recall[3] . . . 24
6.2 Illustration of Replacing Invalid Result, Example: F-C,W2V . . . 25
6.3 Scatter Plot: Construction Method Comparison, Full Data . . . 27
6.4 Scatter Plot: Construction Method Comparison, C-F . . . 28
6.5 Illustration of Data Combination . . . 36

List of Tables

3.1 Notations and Meanings 7

5.1 Dataset Parameters 18
5.2 GraphSAGE parameters 20
5.3 FastGCN parameters 20

6.1 Variations of Experiment Parameters 24
6.2 Construction Method Comparison: All Data 28
6.3 Best Baseline Results Demonstration 29
6.4 Comparison of Baseline Methods: F1 Micro Score 29
6.5 Comparison of Baseline Methods: Difference Between F1 Micro and F1 Macro . . 29
6.6 Best DAGCN Results Demonstration 30
6.7 Comparison of DAGCN Methods: F1 Micro Score 30
6.8 Comparison of DAGCN Methods: Difference Between F1 Micro and F1 Macro . . 30
6.9 Comparison Between DAGCNs and Baseline 31
6.10 Comparison Between DAGCNs and ADDA 31
6.11 Best Oversampling Results Demonstration 32
6.12 The Trend of Oversampling: F1 Micro Score 33
6.13 The Trend of Oversampling: Difference Between F1 Micro and F1 Macro 33
6.14 The Trend of Oversampling: F1 Macro Score 33
6.15 Comparison of Oversampling Methods: F1 Micro Score 34
6.16 Comparison of Oversampling Methods:Difference Between F1 Micro and F1 Macro 34
6.17 Word Embedding Comparison: F1 Micro Score 35
6.18 Word Embedding Comparison: Difference Between F1 Micro and F1 Macro 35
6.19 Comparison About the Data Pairs' Origins 36
6.20 Original Textual Examples of Cyberbully Data,Correct Non-bully 37
6.21 Original Textual Examples of Cyberbully Data,Correct Bully 37
6.22 Original Textual Examples of Cyberbully Data,False Non-bully 38
6.23 Original Textual Examples of Cyberbully Data,False Bully 38

A.1 All Oversampling Cases 49
A.2 Minority Numbers in Oversampling 49

B.1 Bridge Nodes Number and Source-Bridge Edges Degree 52

C.1 Cases Included for Comparison Between No Sampling and Normal Oversampling . 53

D.1 Cases using maGAN and ADASYN Oversampling Methods 54

E.1 Normality Test For All Data Groups 56

F.1 All Experiment Results 77

Chapter 1

Introduction

1.1 Motivation

Cyberbully is defined as "any behavior performed through electronic media by individuals or groups of individuals that repeatedly communicates hostile or aggressive messages intended to inflict harm or discomfort on others"[48]. In other researches, the "repeat" component is taken out of the definition of cyberbully[42]. It is a harmful phenomena that frequently happens in many online social platforms. Various surveys have found that the frequency of prevalence of cyberbully in teenagers ranges from 5% to 55%. Typical cyberbully victims will feel sad, frustrated, angry, anxious, embarrassed and afraid. A significant portion of them have emotional and peer problems. This could lead to poor concentration and depression, eventually inducing low school achievement. They could also undergo clinical symptoms such as headaches, recurrent abdominal pain and problems sleeping. The negative effect of cyberbully is substantial and severe, hence demands method of prevention and remedy,in case it already happened. To automatically recognize cyberbully content among the huge amount of text in social platforms is crucial to the prevention of its occurrence.

The extensive existence of cyberbully phenomena and its conspicuous harm require almost all online social platforms to have cyberbully detection mechanism to prevent or punish these instances. However, not all social platforms have specific data for training a cyberbully detection framework. In fact, most of the cyberbully detection related data are from the most popular platforms, like Twitter, Youtube, Wikipedia, Reddit or Instagram. And generating a new dataset involves human labeling(meaning hiring people to manually classify each instance of data), which is very time consuming and expensive. Even within the same platform, the use of language could change over time, leading to datasets collected from different timestamps to have differences in language usage. The differences may include frequency of words and structure of sentences. Those differences could cause the datasets to emerge like from different platforms. Dataset collected under different topics or different subgroups of users could also have those differences in language usage, despite they might be from the same social platform. it is impossible to develop labeled dataset and train corresponding cyberbully detection classifier for each social platform and its sub-topics and sub group of users, and renew them in a fixed period of time. Consequently, it is very beneficial to have a cross platform cyberbully detection framework that is trained through dataset from one platform and applicable to data from a different platform.

Domain adaptation solves the problem when the probability distributions of source data and target data are different. There are labeled source data and a well-trained classifier for the source domain, and we want the classifier to also work well for the unlabeled target data, which follow different distributions. In cross platform scenario, it is assumed that data from different platforms have their own unique language traits, and these traits result in the different distributions

of embedding vectors that are learnt from the data. Hence, different social platforms, or other conditions that causes different distributions between training data and test data, could be treated as different domains. And when classifier is needed in the domain that only has unlabeled data, the domain adaptation method naturally comes into use—to train classifier in the domain that has labeled data and then adapt the classifier. Figure 1.1 briefly summarizes of reason why cross platform could be regarded as a special case of domain adaptation.

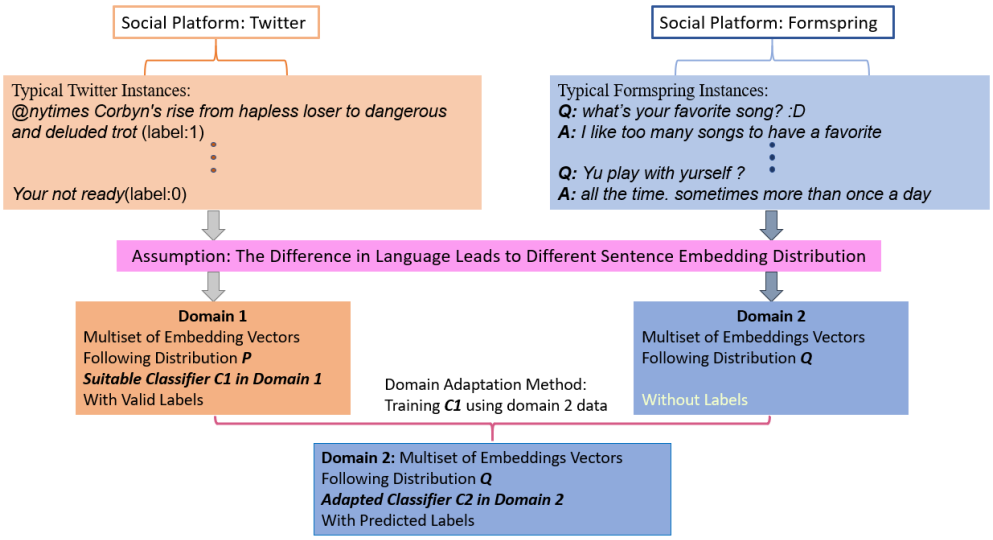

Figure 1.1: Illustration of Relationship Between Cross Platform and Domain Adaptation

1.2 Research Content

Our domain adaptation framework is a graph-based semi-supervised learning method. It was first proposed in [35] and in this project we extend it to cross platform cyberbully detection scenario. In real world, there is significantly more unlabeled data than labeled data. Semi-supervised learning uses labeled data along with an often relatively small portion of unlabeled data to train the classifier, and then performs prediction. Graph-based semi-supervised methods define a graph in which nodes are labeled and unlabeled instances in the dataset, and edges (weighted or unweighted) are usually based on the correlations,e.g. similarity, between the node instances. The predicting of unlabeled nodes could utilize network attributes. (Note that in the following part of this article, the word "graph" and "network" will be used interchangeably)

1.3 Research Questions

This project focuses on using DAGCN to solve cross platform cyberbully detection problem. There are several aspects to explore about DAGCN's application. The principal one is that **if DAGCN is effective in cross platform cyberbully detection problem and has its performance surpassed the existing methods?** We will set up three non domain adaptation baseline methods and one domain adaptation baseline methods to compare with DAGCN. These methods are going to be experimented in different combinations of source-target datasets, with different word embeddings. And the results will undergo statistical tests to see if DAGCN's advantage in performance is statistically significant.

For graph-based semi-supervised learning part of the method, the way of constructing graph

is believed to have great influence on the classification performance. There are two graph construction method proposed for this project, which will be thoroughly introduced in Section 4.2.3 and Section 5.6.1. The second research question is: **Are the two graph construction methods in this project conspicuously different in terms of performance?**. Again, the way of resolving this question is experimenting them in all combinations of source-target datasets and word embeddings, and compare these two groups of results through statistic tests.

One outstanding feature of cyberbully detection training data is the uneven distribution of bully and non-bully instances—usually non-bully instances is much larger in amount. This often causes class imbalance in the classification result, meaning that the non-bully instances in the data have a significantly higher prediction accuracy overall. Ideally the two(or more) classes should be predicted with the same accuracy. The third research question is thus: **Will oversampling the minority class(the bully class) to increase their number help alleviate the class imbalance problem?**. Depending of the amount of samples increased, there will be several orders of oversampling. Higher order means more samples increased. After finishing all the experiments, the no-oversampling result will be compared to first order oversampling result through statistic tests, and first order oversampling result to the second order, and so on.

Two of our three datasets are from the same social platform, Twitter, but one is several years older than the other. They are still treated as different domain despite the same origin of social platform. The research question on this topic is: **If the surmise that data collected with large time discrepancy could be viewed as belonging to different domain?** The answer to the question lies in the comparison of results between two Twitter data pairs and Twitter-Formspring(also a social platform) data pairs—should the results from these two data pairs are better, especially if the results are better for the non domain adaptation methods, then perhaps they ought not to be regarded as being from different domain.

This paper is organized as follows. Chapter 2 introduces the related work in the realm of cyberbully detection, domain adaptation and graph-based semi-supervised learning. Chapter 3 gives the mathematical formulation of domain adaptation, cyberbully detection and the cross platform condition. Chapter 4 discusses the experiment setup, including the detailed introduction to all the procedures in the DAGCN framework. In Chapter 5, we will introduce the parameters used in the experiment. Chapter 6 is a full analysis on the experiment results. Chapter 7 will get to the conclusions to the research questions and the corresponding experiment, as well as the limitation of our work and direction of future work.

Chapter 2

Related Work

Since the method in this project is application of a domain adaptation approach through a graph-based semi-supervised learning method on the field of cyberbully detection, here three realms of related work will be introduced: cyberbullying detection, domain adaptation and graph-based semi-supervised learning.

2.1 Cyberbully Detection

Most of cyberbully detection detection algorithms are supervised learning methods for binary classification, i.e., dividing the data for prediction into "bully" and "non-bully" classes[41]. The complicated nature of cyberbully leads to complicated features used in these algorithms. Four types of features are most common: content-based features, sentiment-based features, user-based features and network-based features. The earliest content-based features is to construct a profanity dictionary and see if the script has profanity word in it. [37] pointed out that real bullying contents may not have profanity word at all while some non-bullying text have. [8] also incorporated person identification module to detect if the text is about a person. [15] includes more features, such as length of the comments, whether usernames containing profanities and non-standard spelling of the words, in addition to profane words in the comments and personal pronouns. Works in [16] counted the ratio of capitalization as features and [19] took race, culture, sexuality, physical appearance, and intelligence related themes as feature.

To extract the sentiment feature from the comments, most of works used emotional lexicons to categorize the polarity of sentiment(positive, neutral,negative). [31] adapted a different way, using Probabilistic Latent Semantic Analysis (PLSA) to get sentiment features. However, [55] found that only 6% of tweets they extracted contained emotional information. Among them, half of the emotion is fear, but in most cases fear is expressed in a joking way. Hence, a detection system based on sentiments alone is not effective because of non-genuine emotions.

User-based features range from age and gender to sexual orientation,race. [12] and [13] showed that considering the gender feature helps improvement the classification performace. But it must be noted that user information is very easy to falsify. Hence, a verification scheme is necessary after these information is collected.

Network-based features include the user's number of friends, uploads, likes, total time spent online, etc. The early work such as [44] used total time online through mobile phone as feature, while the most frequently used network-based feature is the ego-net feature. [24] discovered that the probability of performing cyberbullying is decreased if ego networks has more people and higher interconnectivity. In other works, membership duration, subscriptions[14] as well as followers' numbers[23] are used as network-based features.

While some cyberbully detection algorithms only use content-based features for classification[58], more recent work often utilize multi-modal information. When the comments are about an image, not only is it the combination of two or more type of features mentioned above, but it involves information about the image as well. In addition to content-based features like presence of informal language, use of sexual words and personal pronouns, [45] also incorporated sentiment element inferred from text and information of the image that comments focused on. The Xbully framework proposed by [11] made use of 5 different kinds of features: text, user profiles, relations between users, image related information(number of shares and likes, image labels),timestamp and location of posting the image. It then constructs an heterogeneous network which has 5 different nodes, corresponding to these 5 different features. After decomposing into sub-network and training node embedding for classification, it achieves excellent result for cyberbully detection.

2.2 Domain Adaptation

Domain adaptation is a subfield in machine learning that deals the problem when the distributions of training data and test data are different but the label space is unchanged[28]. To reduce this distribution difference named domain shift, one approach is to manipulate the feature learning of both source and target domain to minimize this domain shift. Many methods used Maximum Mean Discrepancy(MMD) loss for this purpose. MMD computes the norm of the difference between the means of two domains' data. [34] The Deep Adaptation Network (DAN) uses MMD to layers embedded in a reproducing kernel Hilbert space to match higher order statistics of source and target data distributions. Transfer Component Analysis(TCA) method learns transfer component of the two feature space using maximum mean discrepancy. When the data of these two feature space are projected to the subspace of transfer component, the difference in data distributions is reduced and data properties are preserved. Beside using MMD, deep correlation alignment (CORAL) [47] proposed to reduce domain shift by matching the mean and covariance of the two data distributions.

Another major branch of algorithms are to assume some instances from the source domain can also belong to the target domain, and give some weights to these instances. TrAdaBoost[17] uses AdaBoost-based technology to filter out source data that does not fit target data distribution. Then, it trains model using the re-weighted instances from source domain and origin instances from target domain. Bi-weighting domain adaptation (BIW) [53] can projects the feature spaces of two domains into a common coordinate system, and then find the suitable instances from source domain to add into target domain. [56] proposes a metric transfer learning framework ,weights for instances in the source domain are learned in one framework while Mahalanobis distance is learned simultaneously to maximize the intra-class distances and minimize the inter-class distances for the target domain in a parallel framework. [29] introduce an ensemble transfer learning framework. First, there is a weighted resampling method for transfer learning, TrResampling, to resample the data with heavy weights in the source domain in each iteration. TrAdaBoost algorithm is used to adjust the weights. Then, three classic machine learning algorithms, namely, naive Bayes, decision tree, and Support Vector Machine(SVM), are used as the classifiers for the data after TrResampling.

The most popular family of method is adversarial-based method, which adds a domain adversarial loss to the usual class discriminative loss. [49] first proposed to add a domain classifier to predict the binary domain label of all data and encourage its prediction to be half-half. The gradient reversal algorithm (ReverseGrad)[20] directly maximizes the domain adversarial loss by reversing its gradients. Adversarial Discriminative Domain Adaptation is a general framework that subsumes most of its previous adversarial-based algorithms. It involves encoders to get feature representation for both domains and a discriminator to bring the distribution of these two representations closer. Then it applies the classifier trained in source domain to target domain.

2.3 Graph-based Semi-supervised Learning

There are two main branches of semi-supervised learning algorithms: inductive and transductive. Inductive methods is about finding a classification model, whereas transductive provides predictions for unlabeled data without any model. Most transductive methods are graph-based semi-supervised learning method, which is the emphasis here. Graph-based methods usually have three phases: graph construction,graph weighting and inference.[51]

The first graph-based semi-supervised algorithm is graph min-cut[7]. It constructs graph based on k-nearest neighbor or ϵ neighbourhood (connecting data points whose distance is smaller than ϵ). Then, a single source node v_+ is added and connected with infinite weight to the positive data points, and a single sink node v_-, connected with infinite weight to the negative data points. Min-cut is to find a set of edges with minimal combined weight that, when removed, result in a graph in which no paths from the source node to the sink node. All unlabelled nodes in the final graph that are left in the part containing v_+ are labelled positive, and all unlabelled nodes that are in the component containing v_- are labelled negative.

In many cases, it would be better to estimate the probability that an unlabelled data point has a certain label c. One way to do this is to add auxiliary nodes to let the graph conform to Hammersley-Clifford theorem. Then the random variables binding to each unlabeled node corresponds to a Markov random field. [59] proposed to use Gaussian random field to make computation faster. [61] also introduced the label propagation algorithm for estimating label probabilities. It is an iterative algorithm that calculate unlabeled node's label probabilities by propagating the estimated label at each node to its neighbouring nodes based on the edge weights.

For algorithms focusing on graph construction, there are three well-known approaches: ϵ neighbourhood, k-nearest neighbor and b matching. As the graph structure is so volatile when the value of ϵ or similarity measure changes, ϵ neighbourhood is rarely used now. K-nearest neighbor is the most common method. It has two variations:symmetric k-nearest neighbours constructs an edge if node i is in the k-neighbourhood of node j or vice versa, and mutual k-nearest neighbours, in which an edge is constructed if i and j are both in each other's k-neighbourhood. But k-nearest neighbor could lead to severe node unbalance problem: some nodes has much more edges than other nodes, especially when using symmetric k-nearest neighbours. This has bad influence on prediction results[25]. B-matching's goal is to set each nodes to have b degrees and the sum of all edge weights is maximized. Matching refers to the process of trying to find a subset of edges in the graph such that the edges do not share any vertices.

Chapter 3

Problem Statement

Table 3.1 lists all the notations and corresponding meanings used in this chapter.

Notation	Meaning	Notation	Meaning
D_s	Source Domain	x_i	Data Instance in Source Domain
y_i	Label of x_i	R^d	Real Coordinate Space of Dimension d
Y	Label Space	D_t	Target Domain
?	Unknown Label	n_s/n_t	Amount of Data in Source/Target Domain
x^s/y^s	Data/Label From Source Domain	P/Q	Data Distribution of Source/Target Domain
x^t/y^t	Data/Label From Target Domain	G	Classifier for Domain Adaptation
$w_1, w_2...w_n$	Individual Words	C	Classifier for Cyberbully Detection
x^c/y^c	Data/Label for Cyberbully Detection	u_i/y^u	Unlabeled Data/Its True Label
P_1/P_2	Different Social Platform	S_1/S_2	Set of All Words Emerged in P_1/P_2
M_1/M_2	Multiset of All Words Emerged in P_1/P_2	Q_1/Q_2	Discrete Distribution of Elements in M_1/M_2
Q^{P_1}/Q^{P_1}	Distribution of Averaged Word Embedding	x^{P_1}/x^{P_2}	Averaged Word Embedding

Table 3.1: Notations and Meanings

This chapter will introduce the mathematical definitions of cyberbully detection problem, domain adaptation problem and cross platform cyberbully detection problem, and explain why the cross-platform condition could be treated as a variant of domain adaptation.

Problem 1. ***Cyberbully Detection****: In the cyberbully detection problem, each piece of data x_i is a sentence consisting of indefinite number of words(punctuations included sometimes) $x_i = \{w_1, w_2, w_3...w_n\}$(n is not the same for different i). And word w_j can be represented in R^d space. The data x_i has label "0" or "1". "0" means "not bully",sometimes refered to as "non-bully". "1" indicates that it is a bully instance. The key of cyberbully detection is to train a classifier $C : x^c \rightarrow y^c$ using these labeled data(x_i) to correctly give label to previously unlabeled data u_i. u_i is the same type of sentence as x_i, and has a true label y^u unknown beforehand. In reality, cyberbully detection is to make the probability $C(u_i) = y^u$ as high as possible as the classifier may*

not be perfect.

Problem 2. ***Domain Adaptation****: In unsupervised domain adaptation problem, there is a source domain* $D_s = \{(x_i, y_i)\}_{i=1}^{n_s} \subset R^d \times Y$*, in which the total amount of data is* n_s*,* d *is the dimension of the data* x_i *itself ,* Y *is the label* y_i*'s realm. There is also an unlabeled target domain* $D_t = \{(x_i, ?)\}_{i=1}^{n_t} \subset R^d \times Y$ *of size* n_t*. The data in source domain follows the joint distribution* $P = (x^s, y^s)$ *and data in target domain follows joint distribution* $Q = (x^t, y^t)$*. The goal of this project is to extend DAGCN classifier(refered to as G here) so that the classifier* $G : x^t \rightarrow y^t$ *trained through source data is able to make correct prediction* $G(x_t) = y_t$ *with high probability. In this project* $Y = \{0, 1\}$*, with* 0 *representing "non-bully" and* 1 *indicating "bully". The domain adaptation method could also be extended to problems involving multiple classes.*

Problem 3. ***Cross Platform Cyberbully Detection****: In cross platform cyberbully detection problem, we assume that different social platform has different habits in their language. That is, for two social platforms* P_1 *with labeled data and* P_2 *with unlabeled data, the sets of all words in respective platforms are* $S_1 = \{w_1^{(P_1)}, w_2^{(P_1)}, w_3^{(P_1)} ... w_{n_1}^{(P_1)}\}$ *and* $S_2 = \{w_1^{(P_2)}, w_2^{(P_2)}, w_3^{(P_2)} ... w_{n_2}^{(P_2)}\}$*, with* n_1 *and* n_2 *being the total number of different words in each platform. Consider the union of* $S = S_1 \bigcup S_2$ *and let* S_1 *and* S_2 *become multisets* M_1 *and* M_2*, we can have discrete probability distributions* Q_1 *representing the frequencies of all elements in* M_1 *and* Q_2 *representing the frequencies of all elements in* M_2*.* Q_1 *and* Q_2 *are distinct discrete probability distributions that belong to probability space* (S, S_1, Q_1) *and* (S, S_2, Q_2)*. One essential step is to average the word embeddings in data* x_i *to get sentence embedding* s_i*:*

$$s_i = \sum_{k=1}^{n} w_k \tag{3.1}$$

Now the data from the two platforms fall within the space $R^d \times Y$*, they can be represented by joint distribution* $Q^{P_1} = (x^{P_1}, y^{P_1})$ *and* $Q^{P_2} = (x^{P_2}, ?)$*. Here we also assume that the difference in discrete probability distribution* Q_1 *and* Q_2 *lead to the different distribution of* x^{P_1} *and* x^{P_2} *after averaging. Thus, the cross platform cyberbully detection problem becomes the same form as the domain adaptation problem formulated in paragraph 1. So the cross platform cyberbully detection problem can be tackled by domain adaptation method.*

Chapter 4

Framework

In this chapter, there will be introduction to the DAGCN framework and introduction to the framework of the whole project. Framework of the whole project includes a more detailed DAGCN introduction and the data cleaning, data-to-embedding conversion.

4.1 Framework of DAGCN

The original DAGCN has 4 steps:

1. Neural Network Pre-training
2. Bridge Selection
3. Affinity Graph Construction
4. Classification on the Graph

DAGCN in this project will not incorporate neural network pre-training step. And the order of item 2 and 3 is not strictly followed. The bridge selection step requires a source matrix of size $n_s \times d$ and a target matrix $n_t \times d$. Part of the target samples will be chosen as bridge nodes. Affinity graph construction involves in-domain edge construction(source domain and target domain) as well as intra-domain—source to bridge and bridge to target. Classification on the graph is performed through the selected graph based semi supervised classifiers. Figure 4.1 gives a graphic demonstration of structure of DAGCN:

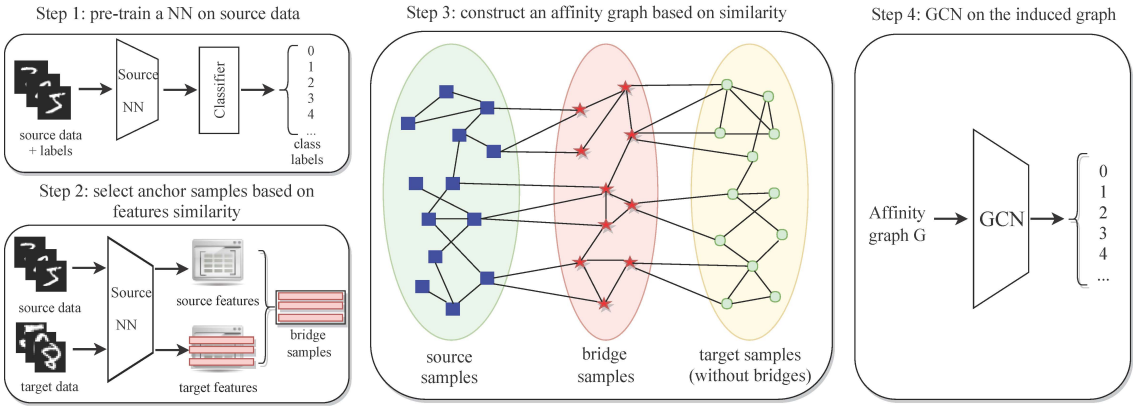

Figure 4.1: DAGCN Structure

There will be a more elaborated explanation in Section 4.2.3 and 4.2.4 as to how the graph is constructed and which classifiers used in the last step.

4.2 Framework of the Project

The framework of this project is divided into 4 parts: cleaning of cyberbully data; converting each instance of data to embedding vectors; constructing graph based on the embedding vectors;and performing classification. Figure 4.2 shows the four steps as well as the production after each of these steps.

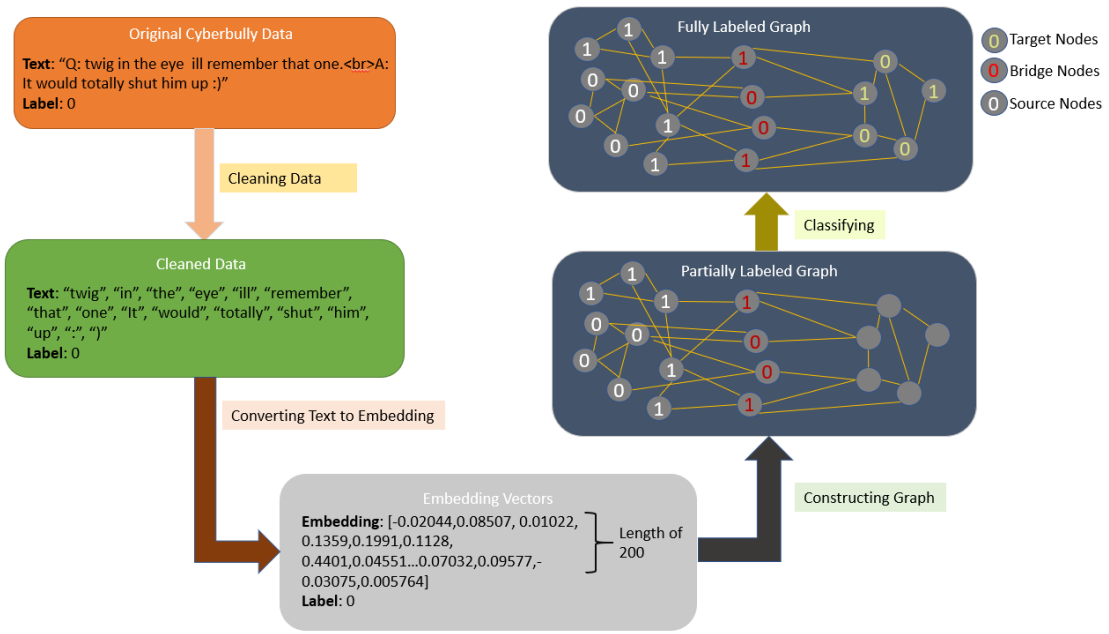

Figure 4.2: Framework Illustration

4.2.1 Data Cleaning

Cleaning the text data is divided into three steps: first is to remove the unwanted content in the original text, such as the punctuation in the sentence when using Word2Vec vocabulary, as Word2Vec vocabulary does not contain the corresponding embedding for punctuation, or some meaningless strings; then is to tokenize the sentence, i.e., separate the individual words inside a sentence; the last step is to lemmatize each word, i.e., map a word's all declension and other type of morphs to its origin and decapitalize the letters.

4.2.2 Sentence to Embedding

Word embedding is a language model that uses high dimensional, real valued vectors to represent words. We use 4 types of embeddings:Word2Vec, Glove(Global Vectors for Word Representation), Bert(Bidirectional Encoder Representations from Transformers) and a Bert based model named Sentence Transformer. Figure 4.3 gives a graphic demonstration of a general way of converting text instances to embedding vectors.

Word2Vec

Word2vec is initially published in [30] in 2013. It improved neural network based langauage models Feedforward Neural Net Language Model(NNLM) and Recurrent Neural Net Language Model(RNNLM) by removing the non-linearity brought by the hidden layer. In Word2Vec model,

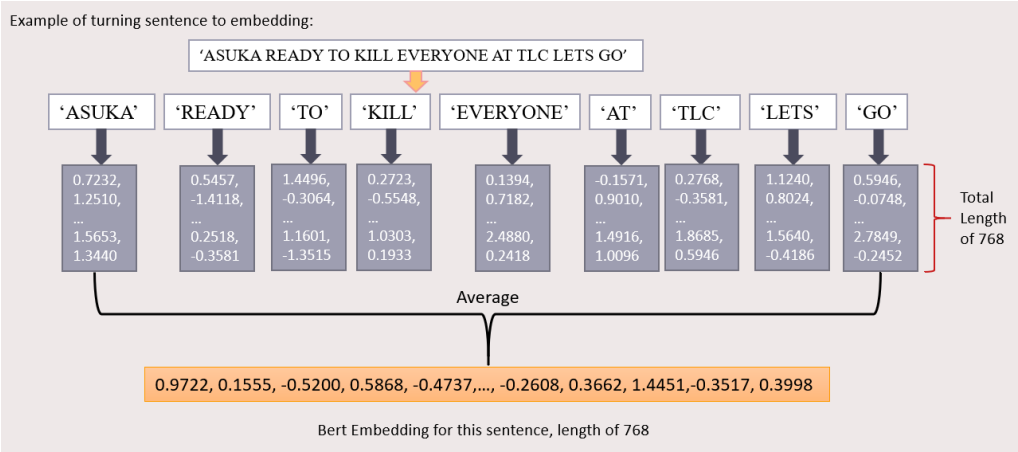

Figure 4.3: Process of Embedding Conversion

the projection layer is shared for all words and their vectors are average. It uses continuous bag-of-words (CBOW) or continuous skip-gram model to get the distributed representation of words, i.e, the high dimensional vectors. In the continuous bag-of-words architecture, the model predicts the current word through a window encompassing the surrounding words. It ignores the order of surrounding words due to the bag-of-words assumption. In the continuous skip-gram architecture, the model uses the current word to predict the surrounding words. The skip-gram architecture weighs nearby words heavier than distant words. Figure 4.4 briefly demonstrates the structure of these two models. CBOW is faster while skip-gram does a better job for infrequent words. It is easily comprehensible as CBOW's bag of words assumption is simpler.

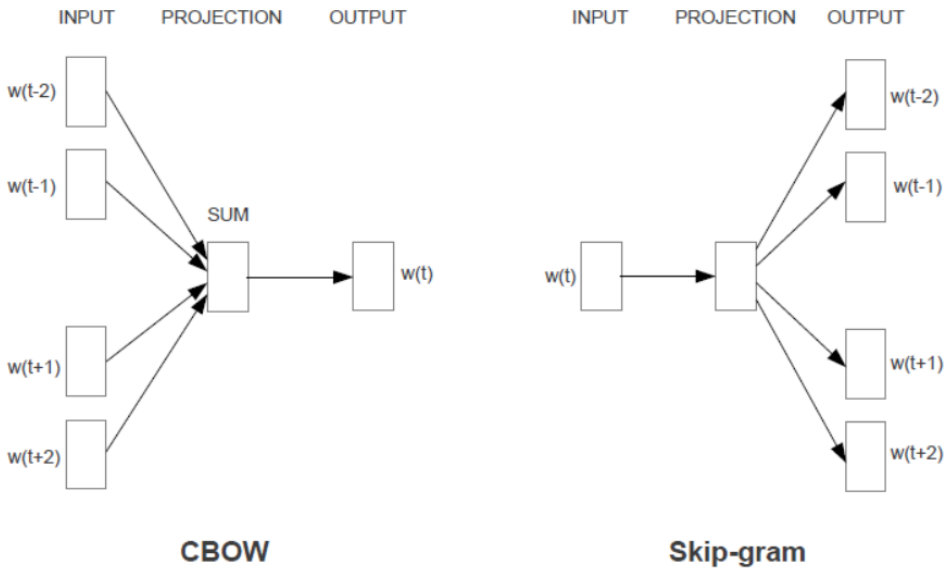

Figure 4.4: Illustration of CBOW and SkipGram model.The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. [30]

Glove

Glove stands for "Global Vectors for Word Representation". It was firstly published in 2016[36].It manages to capture statistical information of a word-word co-occurrence matrix and the context information of the neighboring words(Word2Vec only used the latter). Instead of using the co-occurrence probability directly to evaluate similarity, they pick several context words for the two target words and compare the ratios of conditional probability between these two target words in the presence of a certain context word. If the ratio is close to one, then both the target words are not relevant to the context word. If the ratio is significantly larger or smaller than one, the one of the target word is much closer to this context word than the other. In Figure 4.5, "ice" and "steam" are the target words and "solid","gas","water","fashion" are the context words. It clearly showcases that "ice" is much closer to "solid" than "steam" while "steam" is much closer to "gas" than "ice". They are equally close to "water" and "fashion".

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 4.5: Illustration of Word Similarity Assessment from article [36]

Glove model assesses the similarity between words in the following way: Also, Glove's model trains only on the nonzero elements in the word-word co-occurrence matrix instead of the entire sparse matrix by setting weight to each element of the word-word co-occurrence matrix.

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & if\ x < x_{max} \\ 1 & otherwise \end{cases} \tag{4.1}$$

In the equation, $f(x)$ is the weight, x is the element in the sparse matrix, α and x_{max} are preset values. They use $\alpha = 0.75$ and $x_{max} = 100$. Thus, the zero elements have zero weights and are ignored.

Bert

The full name of BERT is Bidirectional Encoder Representations from Transformers. It was published in 2018[18]. It uses a "masked language model"(MLM) to pretrain a deep bidirectional Transformer(Transformer is a machine learning model published in [52]).After converting each word in the sentence to tokens, the masked language model randomly masks some of the tokens, and trains the model to predict the masked word based only on the context. There are two steps in BERT framework: pre-training and fine-tuning. In the pre-training stage, BERT is trained on unlabeled data from different tasks. For fine-tuning, the model is first initialized with the pre-trained parameters, then all of the parameters are fine-tuned using labeled data from the downstream tasks. A distinctive feature of BERT is its unified architecture across different tasks.

In BERT's input representation, the first token of every sequence input is always a special classification token ([CLS]). Sentence pairs are merged into a single sequence. If the input is a pair of sentence, they are separated with a special token ([SEP]). This scheme is further explained in Figure 4.6.

A learned embedding(Segment Embeddings in Figure 4.6) is added to every token indicating which sentence it belongs to. During the pre-training stage, 15% of tokens in each sequence are masked at random. However, [MASK] token does not appear in fine-tuning. So in practice, if a

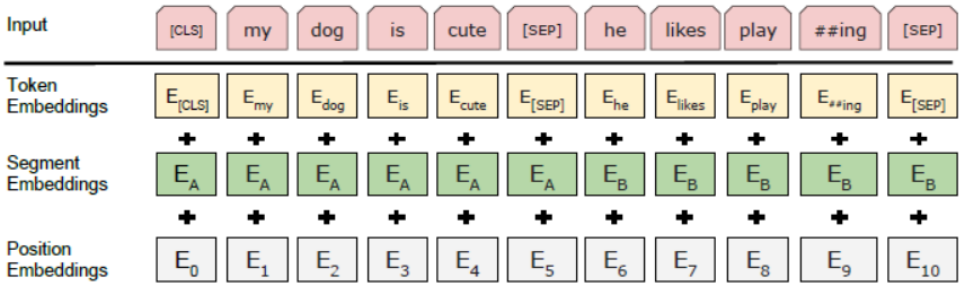

Figure 4.6: Illustration of BERT's Input Representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. [18]

token is chosen as mask, it is replace with the [MASK] token 80% of the time, a random token 10%of the time and itself 10% of the time. In order to let the model catch sentence relationships, BERT is trained with a binarized next sentence prediction task: when choosing sentences A and B as pre-training sample, 50% of the time B is the actual next sentence to A , and 50% of the time B a random sentence from the corpus.

In the fine-tuning stage, a common way for dealing with text pairs is to independently encode them before applying bidirectional cross attention. BERT encodes the concatenated text pair directly, effectively including bidirectional cross attention between two sentences. The acquired token representations are fed into an output layer for token level tasks, such as sequence tagging or question answering, and the [CLS] token's representation is fed into an output layer for classification, such as entailment or sentiment analysis.

Sentence Transformer

Sentence Transformer, originally published in [39],is a model that fine-tunes BERT / RoBERTa / DistilBERT with a siamese or triplet network structure[43] to produce semantically meaningful embeddings for sentences. Sentence transformer adds a pooling operation to the output of BERT to get fixed sized sentence embedding. There are three pooling strategies: Using the output of the CLS-token, computing the mean of all output vectors (MEANstrategy), and computing a max-over-time of the output vectors (MAX-strategy). In the fine-tuning stage, there are three distinct objective functions for siamese and triplet networks: Classification Objective Function, which includes calculating the element-wise difference $|u-v|$ for sentence embeddings u and v and multiplies it with the trainable weight W_t and optimizes cross-entropy loss; Regression Objective Function, which is to calculate the cosine similarity between the two sentence embeddings and optimizes mean squared-error loss; Triplet Objective Function, which means setting up an anchor sentence a, a positive sentence p, and a negative sentence n and optimizes the following triplet loss function:

$$max(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0) \tag{4.2}$$

In Equation 4.2, s_x is the sentence embedding for a, n, p. $||\cdot||$ is Euclidean distance and $\epsilon = 1$ is a preset margin. The embeddings generated by Sentence Transformer are reported to achieve much improvement compared to embeddings acquired from directly averaging BERT or Glove word embeddings.

4.2.3 Network Construction

We use k nearest neighbor(kNN) method to first construct in-domain subgraphs as kNN is the most common graph construction method[51].Then there is connecting the subgraphs of source

domain and target domain. We preset k as 20. Each vector representing the source data is connected to its k nearest neighbor vectors which also represent other source data and share the same label. The same applies to the part of network constructed in target data. But for in-domain subgraph construction, there is no "share the same label" requirement. Then some data from the target domain is selected to form a bridge to connect these two part of the network. Let s_i denote a data point in the source domain and t_i being one from target domain. If for a certain t_i, it has at least m neighbors in the source domain that share the same label,then t_i is a candidate for being bridge node. Once selected, the bridge node t_i will also acquire pseudo label which is identical to their neighbors in the source domain. In the situation that number of bridge nodes candidates is too large, an upper threshold can be set or increase the value of m. In the next stage of semi-supervised learning, these bridge nodes will be treated as labeled data.

4.2.4 Semi-Supervised Learning

The original DAGCN framework uses Graph Convolution Network(GCN)[27] classifier for the semi-supervised learning step. We use three graph-based semi-supervised learning method:GraphSAGE, FastGCN, and an adapted version of FastGCN—FastGCN plus a discriminator—for the classification task. GraphSAGE and FastGCN are both derived from GCN with some improvements.

GraphSAGE

The full name of GraphSAGE framework is Graph Sample and Aggregate[21]. Based on the GCN approach, GraphSAGE uses aggregator function to learn the topological structure of each node's neighborhood and the information of its neighbor's features. Then it comes up with a embedding function that is also applicable to unseen nodes. Figure 4.7 shows how GraphSAGE algorithm works.

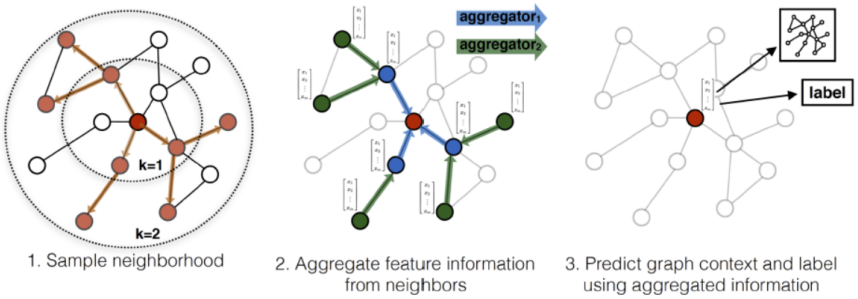

Figure 4.7: Illustration of GraphSAGE's Sampling and Aggregation from [21]

GraphSAGE model parameters can be learned through standard stochastic gradient descent and back-propagation algorithms. And it uses forward propagation for embedding generation. When sampling neighbors, GraphSAGE defines a fixed size of each node's neighborhood in order to keep constant batch size. Three types of aggregators are studied: Mean aggregator, which takes element-wise mean of the vectors of neighbor's features and is invariant to order of neighbors(symmetric); LSTM aggregator, which has larger expressive capability but arranges a random permutation of the node's neighbors due to asymmetry; Pooling aggregator, in which each neighbor's vector is independently fed through a fully-connected neural network and, an element-wise max-pooling is applied. The pooling aggregator is also symmetric.

FastGCN

One of the major improvement FastGCN[9] over GCN is also to relax the necessity for all data to be present to learn node embeddings. Training algorithm like Stochastic Gradient Descent(SGD) and its batch generalization requires independent data samples. However, for graphs, each vertex

is convolved with all its neighbors and hence loses independence. FastGCN assumes that there is a larger graph G' with vertex set V' in the probability space $(V'; F; P)$. For the given graph G, it is an induced subgraph of G' and its vertices are *iid* samples of V' conforming to the probability measure P. FastGCN also holds that each layer of GCN defines an independent embedding function of the vertices. The embedding functions from two consecutive layers are related through convolution, expressed as integral transform. Figure 4.8 explains how the approximation of graph convolution through integrals works. In the left part(graph convolution view), each dot represents a node in the graph . For two consecutive rows, a dot i in one row is connected in gray line to dot j in the other row if they are connected in the graph. The convolution layer mixes the node embeddings based on the graph connectivity features. Nodes are subsampled through bootstrapping in each layer to approximate the convolution. The sampled portions are denoted by the solid blue dots and the orange lines.In the right part is the integral transform view. The embedding function in the next layer is an integral transform (illustrated by the orange fan-out shape) of the one in the previous layer.

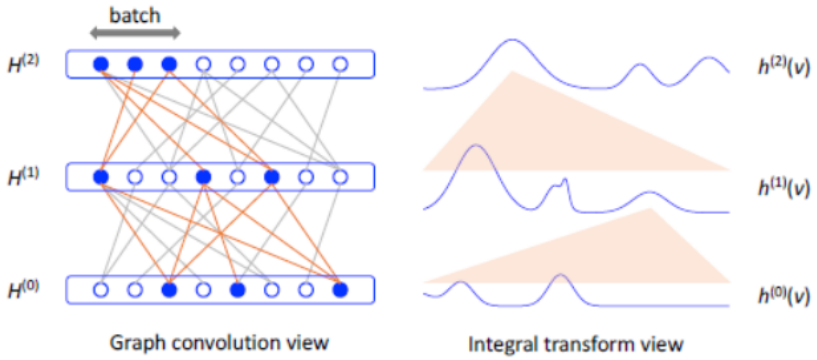

Figure 4.8: Illustration of FastGCN's Integral Estimation from [9]

Now the integrals can be evaluated by the Monte Carlo method, which enables batched training algorithm and a separation of training and test data and makes the algorithm run at a faster speed.

FastGCN with Discriminator

The goal of adding a discriminator is to bridge the distribution gap between source embedding vectors and target embedding vectors. This could be done before graph construction, as edges' weights are determined by embedding vector's similarity so embedding vector can no longer be changed after constructing graph. [46] is an example of bridging the distribution gap by aligning their second-order statistics. The FastGCN architecture in this project has two layers,and the first one is a fully connected layer. The outcome of this layer is the intermediate embedding vector of the same size as the original embedding vector. The intermediate embedding vectors could still belong to two different group with distinct distribution. The discriminator is able to help bridge the possible distribution gap the intermediate embedding vectors.

The input of discriminator is the original embedding vector with domain label—"0" meaning from source domain and "1" indicating target domain. The discriminator has two fully connected layers: the input of both layers and output of the first layer are all of the same dimension as the input embedding vector, and the output dimension of the second layer is 2. The discriminator will predict the domain label for the intermediate embedding vectors. If the discriminator is completely unable to tell which domain the node embedding is from, then the distribution gap has been filled in intermediate embedding vectors. Suppose being label "1" is positive and being labeled "0" is

negative, then the discriminative loss is defines as:

$$Discriminative\ Loss = \frac{1}{2}(\frac{|TP - FN|}{TP + FN} + \frac{|TN - FP|}{TN + FP}) \tag{4.3}$$

TP,TN,FP,FN implicate true positive, true negative, false potive, false negative respectively. The best case is the discriminator cannot distinguish the domain origin of the node embeddings. When it happens, half of the originally "positive" instances will be predicted positive and the other half negative, so do the originally "negative" instances. In this case, the loss is equal to zero. The worst situation is that when the origin domain of all instances are correctly predicted or falsely predicted. In that case the mixing attempt of the domain origin is completely failed. Hence the loss reaches its maximum, which is 1. The discriminative loss will multiply a coefficient and add to the FastGCN loss. The detailed structure of the discriminator is introduced in Chapter 5, Experiment Setup.

4.2.5 Time Complexity Analysis

For our algorithm, the conversion from text to embedding has time complexity $O(N)$(N is the number of nodes in the constructed graph, which is also the total number of instances in both source and target dataset). Calculating sorted similarities matrices inside source domain, inside target domain and between these two domains to select bridge nodes in total cost $O(N^2logNK)$ time. Here is a brief explanation: Let S denote number of data in the source domain ,T represent the number in target domain ,B denote the number of selected bridge nodes and K is the length of embedding, computing two in-domain similarity sorted matrix takes S^2logS and T^2logT time and one cross domain similarity matrix takes $STlogS$ time. $N^2logN = (S^2 + T^2 + 2ST)logN > (S^2 + T^2 + ST)logN > S^2logS + T^2logT + STlogS$, so in total it costs $O(N^2logN)$ time. Taking the embedding length K into account, we get time complexity $O(N^2logN)K$. Constructing source-source and target-target edges with the help of previously calculated sorted matrices has time complexity $O(mN)$ and $O(kB)$. According to [62], the time complexity for GraphSAGE is $O(bKs^L_{node} + bK^2s^{L-1}_{node})$ and for FastGCN is $O(LKs^2_{layer} + LK^2s_{layer})$. b is batch size. L is the number of GCN layers. s_{node} is the number of sampled neighbors per node for node-wise sampling, and s_{layer} is number of sampled nodes per layer for layer-wise sampling. In our case, batch size b for GraphSAGE is 128. L equals to 2, s_{node} is 20 for the first layer and 8 for the second layer. For FastGCN, $s_{layer} = 256$. For GraphSAGE, the time complexity is for one batch. The time complexity for running all batches should be $O(NKs^L_{node} + NK^2s^{L-1}_{node})$. Since all the parameters from GraphSAGE and FastGCN algorithms are significantly smaller than the total nodes N, as none of the unique parameters from GraphSAGE and FastGCN is larger then 200 while N is larger than 10000, the overall time complexity of our algorithm is $O(N) + O(N^2logNK) + O(bKs^L_{node} + bK^2s^{L-1}_{node}) = O(N^2logNK)$ or $O(N) + O(N^2logNK) + O(LKs^2_{layer} + LK^2s_{layer}) = O(N^2logNK)$.

Chapter 5

Experiment Setup

In this chapter, we will give a detailed introduction to our experiment settings. It includes the introduction to the dataset we used, the method for converting text in the dataset to embeddings, the selection of baseline method for comparison, the oversampling for source data and the parameters for network construction and semi-supervised learning approaches.

5.1 Goal of Experiment

The goal of the experiments is to explore the aspects mentioned in the research questions. Each experiment has 5 variables: source-target data pairs, embedding, oversampling, graph construction method, classifier. By allocating result data into different groups based solely on one variable and compare these groups through statistic tests, we can evaluate the choices under this variable. Specifically,

1. **Classifier**: Dividing result data into groups by the "classifiers" variable enables comparison between different classifiers, thus if DAGCN classifiers have better performance than baseline classifiers.
2. **Graph Construction Method**: Telling if graph construction method yields different results requires result data from DAGCN classifiers to be separated by graph construction method.Then do the analysis through statistic tests.
3. **Oversampling**: Dividing result data by different oversampling schemes could illustrate if oversampling reduces class imbalance problem.
4. **Source-target Pairs**: Dividing result data by source-target data pairs and comparing the results from two Twitter data pairs to others answers if the two Twitter dataset can be regarded as two domains.
5. **Embedding**: Dividing result data by its embedding methods shows which embedding is better.

The first 4 cases correspond to the research questions in Section 1.3.Although the aspect of embedding method is not mentioned in the research questions, its analysis could help reduce the number of experiments in the future work.

5.2 Dataset

we give a brief introduction to the 3 datasets used in the experiment. Table 5.1 shows some of their information.

The first dataset is used in the [10]. It has 19993 pieces of comments collected from Twitter, within which there are 3845 piece marked as "bully". This originally unnamed dataset is referred to as "CBASU"(meaning "Cyberbully data from Arizona State University") and later "C" in the experiment record chart.

The second dataset[40] has 12698 pieces of comments collected from Formspring, a online question-answer social platform which was merged into other websites since 2015. This dataset is collected in the year of 2010. Each piece in this dataset is decided to be bully or not by three independent annotator. The rule here is an instance is considered bully only when two or more annotators agree. So there are 773 pieces of comments marked as "bully".It is referred to as "F" in the experiment record chart.

The third dataset[57], OLID, has 13240 instances collected from Twitter, in which there are 4400 bully instances. OLID stands for "Offensive Language Identification Dataset", which is originally designed to classify cyberbully types. Unlike the other two dataset, which are collected several years before, OLID dataset is collected around year 2018.

Dataset Name	**Abbreviation**	**Size**	**Bully Instances**	**Bully Ratio**	**Source**
CBASU	C	19993	3845	19.23%	Twitter
Formspring	F	12698	773	6.09%	Formspring
OLID	O	13240	4400	33.23%	Twitter

Table 5.1: Dataset Parameters

5.3 Data Cleaning&Embedding Conversion

The method of data cleaning depends on the specific dataset and to which pretrained embedding will it be converted. OLID dataset contains "@USER" string because it is collected from Twitter and we need to remove this particular string. Similar case happens in Formspring dataset. Every piece of data contains special string "Q:" and "A:", which stand for "question" and "answer" as Formspring is a question-answer platform. As mentioned before, punctuations need to be removed for Word2Vec. And when using SentenceTransformer to get embeddings, we do not need to do any further cleaning. Otherwise,the next is to tokenize the sentence and lemmatize each word. To get the embedding, we look for each lemmatized word(or punctuation) in the pretrained embedding's library and average all words' embedding as the sentence's embedding. Some word has more than one corresponding embedding in the pretrained Bert vocabulary, and we just average all these embeddings.

5.4 Baseline Methods

We use support vector machine(SVM) with polynomial kernel(poly) and Radial Basis Function(RBF) kernel, as well as random forest method as non domain adaptation baseline. The degree for the polynomial kernel is 5. Other parameters of both SVM classifiers are in accordance with the default Python function sklearn.svm.SVC[6]. The parameter for the random forest classifier is: number of estimators=600 ,max depth=15. Other adjustable parameters are in accordance with the default sklearn.ensemble.RandomForestClassifier[1] function of Python package sklearn.

The domain adaptation method for comparison is Adversarial Discriminative Domain Adaptation
(ADDA)[50]. The general framework of ADDA in our scenario is shown in Figure 1. The first

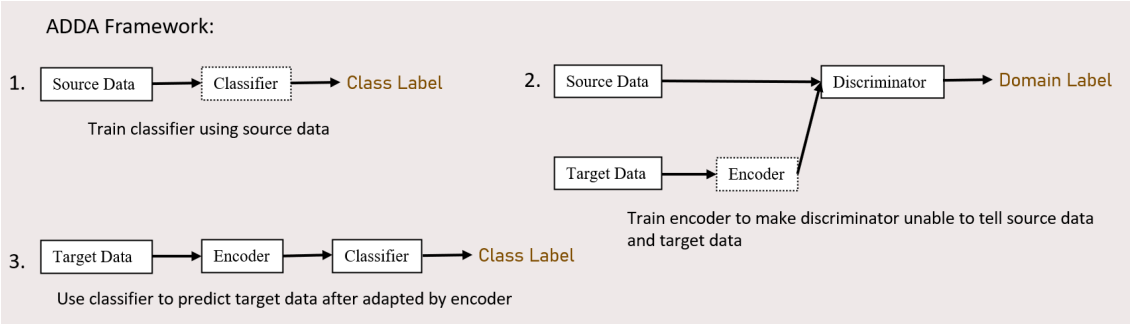

Figure 5.1: ADDA Framework

stage is to train a classifier using the data from source domain. Then we train an encoder for target data. After a successful training, the discriminator can no longer tell a piece of data is from source domain or encoded target domain. In the final stage, the classifier will work on the encoded target data.

The detailed structures of classifier, discriminator and encoder are illustrated in Figure 5.2. The discriminator and classifier have the exactly same structure. The embedding size is 200, 300 or 768 depending on the word embedding used. The output of dimension 2 in the final layer of classifier gives the binary classification result. The classifier,encoder, discriminator are all trained for 150 epochs. The optimizer for training process is Adam optimizer and loss is cross-entropy loss. Batch size is 150 and learning rate is 1e-4.

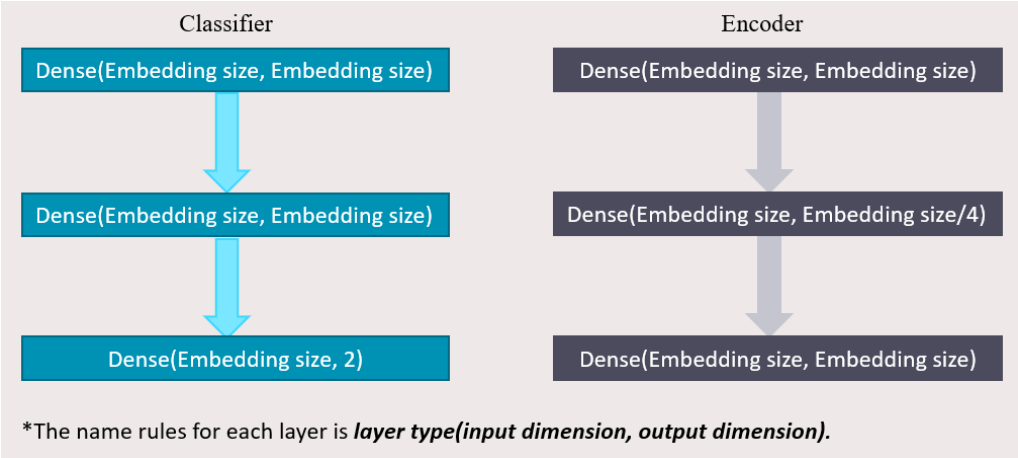

Figure 5.2: Structure of Classifier and Encoder

5.5 Oversampling Schemes

We use three different sampling method to increase the number of minority class in the source data. We hypothesize that this approach could reduce the performance gap between majority class and minority class in the target dataset. These four methods are Naive Random Over-sampling, Adaptive Synthetic(ADASYN)[22] and Generative Adversarial Network(GAN). The first three are provided by Python package Imbalanced-learn[5], and Margin Adaptation for GAN(maGAN)[54] for the last one. For the CBASU dataset, the majority over minority ratio is 4.2, so the number of minorities will be oversampled to 40%,60%,80%,100% of the majority group. The Formspring

dataset's majority over minority ratio is 15.4, so the number of minorities will be oversampled to 10%,20%,30%,40% of the majority group. The OLID dataset's majority over minority ratio is 2.0, so the number of minorities will be oversampled to 60%,80%,100% of the majority group. Note that the oversampling schemes are different when the embeddings or source-target pairs are different. All oversampling cases are shown in the Appendix A. In addition, there is a list displaying the minority class number and the ratio of oversampled minority over majority in the appendix. The description of sampling parameters consists of two parts: sampling method "Normal","ADASYN","maGAN"; percentage, for example, "50P" meaning the minority class is oversampling to 50% of the majority class.

5.6 Implementation Details

5.6.1 Network Construction

We tried two different network construction approaches. The first approach is to let the ratio between number of bridge nodes and number of target nodes approximately between 1:5 and 1:4. To keep the proportion of bridge nodes in target data, the number of source nodes connecting to each bridge nodes varies from 8 to 67. In Appendix B, there is a list demonstrating the exact bridge number and source-bridge edges per bridge node, as there is no unified equation for each particular case. [60] suggests that for graph construction through k nearest neighbor method, usually less edges lead to better performance. In the first approach, sometimes edge numbers could be really large. The second approach is to set bridge nodes to a fixed number, 2098, and the number of souce-bridge edges per bridge node to 5. Each source node is connected to 20 of its nearest neighbors in the source domain with the same label. Each node in the target domain, including bridge nodes, is connected to 20 of its nearest neighbors in the target domain. In two situations, however, the only possible way to construct a reasonable graph is to let k equal to 3 and number of bridge nodes be 47(and 57, in another case).

5.6.2 Semi-supervised Learning

After constructing network, we use GraphSage, FastGCN and an adapted FastGCN classifier to perform semi-supervised learning to see which one has the better performance. GraphSage is run with a LSTM based aggregator. Table 5.2 shows the parameters for GraphSAGE classifier.

Parameters	**Learning Rate**	**Epochs**	**Dropout**	**Weight Decay**	**Samples Layer1**	**Samples Layer2**	**Batch Size**
Values	0.002	30	0.5	0.005	20	8	128

Table 5.2: GraphSAGE parameters

We use mixed layer structure for FastGCN classifier. The detailed parameters are shown in table 5.3.

Parameters	**Learning Rate**	**Epochs**	**Dropout**	**Weight Decay**	**Batch Size**
Values	0.002	150	0.5	0.0004	256

Table 5.3: FastGCN parameters

The loss for FastGCN and GraphSAGE is cross-entropy loss and optimizer is Adam optimizer. The mixed layer structure is demonstrated alongside the discriminator, which we added in the adapted FastGCN. The discriminator tries to tell the intermediate embedding between two layers if they are from the source domain or the target domain, and adds the discriminative loss to the total loss. Thus in theory, we bring the difference between source domain and target domain in

intermediate embedding to minimum. Figure 5.3 shows the details of FastGCN classifier with

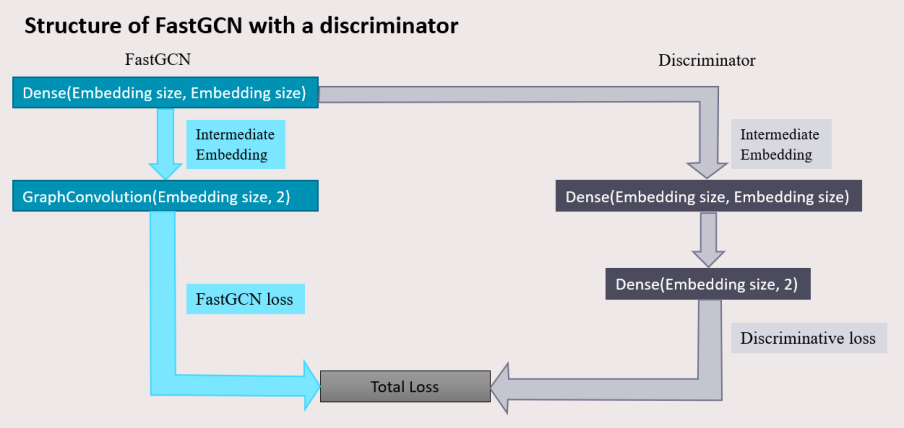

Figure 5.3: FastGCN Structure with Discriminator

a discriminator. The first layer of FastGCN is a dense layer, with both input size and output size being the embedding length(200,300 or 768, depending on the word embedding).The output of this layer goes to the GraphConvolution layer and produces the classification result. The output of the FastGCN's dense layer is the discriminator's input. The total loss is the prediction loss of FastGCN plus the prediction loss of discriminator times 0.5. The FastGCN loss is also softmax cross entropy loss. The discriminative loss is defined in Section 4.2.4. Other parameters of FastGCN with discriminator are the same as FastGCN in Table 5.3.

5.7 Computer System&Hardware

All experiments are run on MSI GP63 leopard 8RE laptop, with CPU being Intel Core i7-8750H, GPU being GeForce® GTX 1060 with 6GB GDDR and 8GB memory. The Python codes are run in Jupyterlab 1.1.4 on Anaconda.

Chapter 6

Result Analysis

In this chapter, we will answer the four research questions proposed in Section 1.3 by comparing results from the experiment. First there is introduction to the statistic tests, the correction method for multiple testings, the meaning of result metrics, and explanations of how the preparation for the result data is done. Then, the comparison of different groups of data and analysis are presented. At the end, there will be individual case analysis which will present some typical instances that are classified as bully or non-bully, as well as typical instances that are falsely classified as bully or non-bully.

6.1 Preparation

6.1.1 Introduction to Sign Test

Given a group of paired data, the sign test can determine if a member of a random pair tends to be larger than the other member of the pair. Many statistic tests have distribution requirements on the data, such as conforming to normal distribution, which is required by paired t test; or equal variances between data groups, which is required by analysis of variance(ANOVA); or paired data follow the same distribution, which is prerequisite for Wilcoxon signed-rank test. Our result data may not satisfiy those requirements. Sign test only requires dependence(or can be paired) for the two groups of data and independence of instances inside data groups, which the results from this experiment could meet.

The computation of sign test's statistics is as follows: suppose there are two groups of same-sized data X and Y, each has data $x_1, x_2...x_n$and $y_1, y_2...y_n$ that can be ordered and their difference $z_1 = x_1 - y_1, z_2 = x_2 - y_2...z_n = x_n - y_n$ are mutually independent. Then the values of z are counted: if $z = 0$ it is discarded; if $z > 0$ then N_+ is added 1;if $z < 0$ then N_- is added 1. Initially N_+ and N_- are set to zero. Sign test's statistic M is calculated by $M = \frac{N_+ - N_-}{2}$.

The null hypothesis H_0 for sign test is that **given paired data (X, Y), one member of a random pair is not larger or smaller than the other member of the pair**. In our analysis, one sided sign test is preferred as we want to know which group specifically has the better performance. So the alternative hypothesis H_1 is **a random sample x_i from X tends to be larger than its pair y_i in** The probability p is calculated by:

$$p = \sum_{k=N_+}^{N_+ + N_-} \frac{(N_+ + N_-)!}{N_+!N_-!}(0.5)^k(0.5)^{N_+ + N_- - k} \tag{6.1}$$

Then the result p value is compared to the significant level 0.05 to determine if the null hypothesis is rejected. The p value means the highest probability of falsely rejecting null hypothesis that we

can tolerate—even if the null hypothesis is rejected, it could still be true, but the probability of null hypothesis being true is less than p value.

Sign test only records the ordinance of the data pairs and ignores the extent of their difference. Hence, the data used for sign test is adjusted to 2 decimal precision instead of originally 4 or 5 decimal. Because the accuracy of algorithms often fluctuates in a range, too small a difference does not imply that there exists a systematical difference.

6.1.2 Introduction to Holm-Bonferroni Method

We use Holm-Bonferroni Method to correct for p values in multiple comparisons. When performing several statistic tests simultaneously, more inferences will increase the chances that erroneous inferences occur. If there are m independent comparisons, then the probability $\bar{\alpha}$ of at least falsely rejecting one null hypothesis is given by:

$$\bar{\alpha} = 1 - (1 - \alpha_i)^m \ , i = 1, 2, ...m \tag{6.2}$$

This $\bar{\alpha}$ is called family-wise error rate. Apparently the value of $\bar{\alpha}$ rises with m. The Holm-Bonfferoni method is to ensure that family-wise error rate is no higher than prescribed α value(0.05 in our case). The method is as follows:

1. Suppose there are m p-values from m independent statistic tests and the corresponding hypotheses $H_1, H_2, ..., H_m$, first step is to sorted the p values in the order of lowest-to-highest and get $p_1, p_2, ...p_m$.
2. If $p_k < \frac{\alpha}{m+1-k} (k = 1, 2..., m)$,then reject hypothesis H_k and let $k = k + 1$. If H_k cannot be rejected, then the rest hypothesis $H_{k+1}, H_{k+2}...H_m$ will not be rejected and the calculation could end at k.

Here is a brief proof for the correctness of Holm-Bonferroni Method[4]. Let I_0 be the set of indices corresponding to the (unknown) true null hypotheses, having m_0 members and h be the serial number of first (incorrectly) rejected true hypothesis (first in the ordering given above), then the hypotheses before h, $H_1, H_2...H_{h-1}$ are all rejected and $h - 1 \leq m - m_0$. So $\frac{1}{m-h+1} \leq \frac{1}{m_0}$. H_h being rejected means $p_h \geq \frac{\alpha}{m-h+1}$. And $\frac{\alpha}{m-h+1} \geq \alpha m_0$. Hence, if wrongly rejecting a true hypothesis happens, there has to be a true hypothesis with corresponding p value at most $\frac{\alpha}{m_0}$. Define a random variable $A = \{p_i \leq \frac{\alpha}{m_0} \ for \ i \in I_0\}$, then we get $Pr(A) \leq \alpha$ by the Bonferroni inequalities[2]. In the tables displaying the results afterwards, p values after Holm-Bonferroni correction will belong to the entry named "p Value(HB)".

6.1.3 Introduction to Experiment Variations and Result Metrics

For each experiment, there are 5 distinct parameters that distinguish it from other experiments: source-target data pairs, embedding method, oversampling,graph construction method and type of classifier. Table 6.1 gives an intuitive view of the variations of these parameters.

**: Full list of oversampling is presented in Appendix A*
***:FastGCN,D** is short of FastGCN with discriminator*
****:Details of these two methods are introduced in 5.5.1*

We use f1 score as the metric for our experiment result. Two kinds of results are recorded in each experiment: f1 score, micro average and f1 score, macro average. Most of the time they are referred to as "f1 micro/macro score" or simply "f1 micro/macro". Below we will first introduce the concept of precision and recall, then f1 score and how the micro/macro average is done.
Precision is defined by $Precision = \frac{TP}{TP+FP}$, and TP stands for true positive whereas FP is false positive. True positives are the instances originally belong to "positive" class and are catalogued

Source-Target Data Pairs	**Embedding**	**Oversampling**	**Graph Construction**	**Classifier**
CBASU-Formspring	Word2Vec	*	1***	GraphSAGE
CBASU-OLID	Glove		2***	FastGCN
OLID-Formspring	Bert			FastGCN,D**
OLID-CBASU	Sentence Transformer			SVM,poly
Formspring-OLID				SVM,rbf
Formspring-CBASU				Random Forest
				ADDA

Table 6.1: Variations of Experiment Parameters

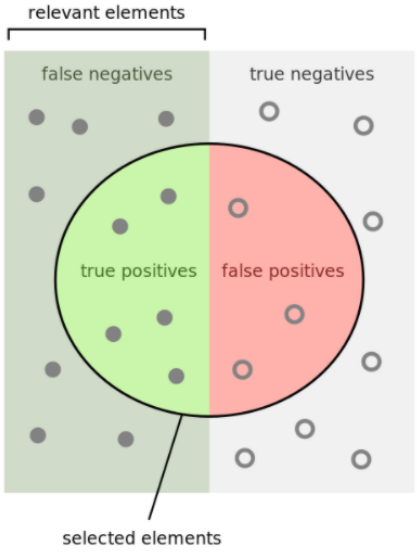

Figure 6.1: Auxiliary Illustration of the Concept of Precision and Recall[3]

in "positive". False postives are those originally belong to "positive" class but catalogued as "negative". Recall is defined as $Recall = \frac{TP}{TP+FN}$. FN refers to false negative, which are the instances originally belong to "negative" class but fall in "positive" class. Then f1 score is calculated by $F1 = 2\frac{Precision \times Recall}{Precision+Recall}$.

For the binary classification case, let TP_1 refers to the true positives in class 1. Since there are only two classes, the TN_1(true negatives for class 1) is the true positives for class 2. The way of micro-averaging precision for binary classification is $Precision_{micro} = \frac{TP_1+TP_2}{TP_1+FN_1+TP_2+FN_2}$. Similarly, micro-averaging recall is $Recall_{micro} = \frac{TP_1+TP_2}{TP_1+FP_1+TP_2+FP_2}$. Note that FP_1 is equivalent to FN_2 and FP_2 is equivalent to FN_1, hence $Precision_{micro} = Recall_{micro}$. So

$$\begin{aligned} F1_{micro} =& 2\frac{Precision_{micro} \times Recall_{micro}}{Precision_{micro} + Recall_{micro}} = Precision_{micro} = \frac{TP_1 + TP_2}{TP_1 + FN_1 + TP_2 + FN_2} \\ &= \frac{TP_1 + TN_1}{TP_1 + FN_1 + TN_1 + FP_1} = Accuracy \end{aligned} \tag{6.3}$$

Thus we show that how F1 micro average score is calculated, and it is equal to accuracy. For macro average, $Precision_{macro} = \frac{1}{2}(\frac{TP_1}{TP_1+FN_1} + \frac{TP_2}{TP_2+FN_2})$ and $Recall_{macro} = \frac{1}{2}(\frac{TP_1}{TP_1+FP_1} + \frac{TP_2}{TP_2+FP_2})$.

And f1 macro average score is calculated by:

$$F1_{macro} = 2\frac{Precision_{macro} \times Recall_{macro}}{Precision_{macro} + Recall_{macro}} \tag{6.4}$$

6.1.4 Processing Invalid Results

Some results are deeply disrupted by the class imbalance problem. There are situations that all(sometimes almost all)instances are predicted as one class, while none or almost no instances are predicted as the other class. These results are considered invalid. Also, if the prediction result contains too few minority class, it is regarded as invalid as well. The "too few" standard is set to 40, which means if a result has less than 40 instances predicted as minority class, it will be treated as invalid as well. The threshold of 40 is chosen based on our subjective observation of the experiment results, so that not too many results are considered invalid. There are two most probable explanations for this phenomena: one is that the classifier is too complex, e.g. overfits, so that it can classify a very small portion of data to one class while the big majority in another class; the other explanation is that the classifier is too simple, and its decision boundary is far from the bulk of the whole data. Both cases lead to undesirable classifier so the result is treated as invalid. In the following comparison sessions, invalid results will be nullified: f1 micro score is set to 0, as it is the worse case of accuracy; f1 macro score is set to 1 to let the difference between f1 micro and f1 macro reach its maximum, which is the worst scenario. Figure 6.1 shows how the replacement is done.

	F1 Micro	F1 Macro
GraphSAGE1	0.82141	0.54956
GraphSAGE2	0.79508	0.44728
FastGCN1	0.80786	0.45478
FastGCN1d	0.80573	0.47161
FastGCN2	0.80512	0.64048
FastGCN2d	0.80502	0.44599
SVM Poly	**0.83130**	**0.59382**
SVM Rbf	0.81131	0.48297
Random Forest	0.80649	0.45509
ADDA	0.8051	0.4460

Replace Invalid Results

	F1 Micro	F1 Macro
GraphSAGE1	0.82141	0.54956
GraphSAGE2	0.79508	0.44728
FastGCN1	0.80786	0.45478
FastGCN1d	0.80573	0.47161
FastGCN2	0	1
FastGCN2d	0.80502	0.44599
SVM Poly	**0.83130**	**0.59382**
SVM Rbf	0.81131	0.48297
Random Forest	0.80649	0.45509
ADDA	0	1

Figure 6.2: Illustration of Replacing Invalid Result, Example: F-C,W2V

Not in all situations can the network be constructed in the two designated ways. When using Glove embedding and the source data is OLID, regardless of which target dataset is, the graph could only be constructed through one way because the bridge-source degree is less than 5. In this case, the result from the first graph construction method is copied to the second graph construction method so that their results are the same. Thus this part bears no weight in the graph construction method comparison. All results data are presented in the table of Appendix F, in which the underlined data are considered invalid.

6.1.5 Normality Test for All Data

Normality Test Results

To prove that the result data indeed do not meet the normality requirement, each group of data will be examined by the Shapiro-Wilk test for normality. Note that invalid results have already been substituted before doing this normality test. It has been proved that Shapiro-Wilk test has

the biggest statistic power among the common normality tests[38]. The null hypothesis for this test is that the data was drawn from a normal distribution. And the threshold for rejecting null hypothesis is 0.05.

The list of results is placed in Appendix E, as it is too long. From the table, it is shown that for most(39 out of 44) of the Shapiro-Wilk test performed, the consequential p values are less than 0.05. Therefore, most of the result data are not normal distributed. Thus, Relatively more powerful parametric test, such as paired t test, cannot be used for analysing results in this project.

Data Formation and Usage

This small section will introduce how these data in Table E.1(table of normality test results) are selected and combined from the long table in Appendix F, and which section below will them be used for comparison.

The first four rows in Table E.1 "All Data for Graph Construction Method" are used in Section 6.2 for graph method comparison—method 1 comparing to method 2, micro to micro, macro to macro. In the example of "All Data for Graph Construction Method 1, F1 Micro", results are first combined by the order of "Graphsage1"-"FastGCN1"-"FastGCN1d" in each variation of source-target pair and embedding method, then combined by word embedding method "Glove"-"W2V"-"ST"-"Bert", then combined by source-target pair "CF"-"CO"-"OC"-"OF"-"FC"-"FO". The rest three are formed in a similar matter.

From the 5th row to the 24th row, there are f1 micro and macro results obtained by SVM-poly, SVM-rbf, random forest, ADDA, GraphSAGE1, GraphSAGE2, FastGCN1, FastGCN1d, FastGCN2 and FastGCN2d. They are first selected from the no-oversampling experiments by the respective method, then combined by word embedding method "Glove"-"W2V"-"ST"-"Bert", then combined by source-target pair "CF"-"CO"-"OF"-"OC"-"FO"-"FC". These results are used in Section 6.3, which is the comparison between DAGCN methods and baseline methods.

25th to 32nd rows of Table 6.2 are the result data for word embedding comparison in Section 6.5. These data also excludes the instances with oversampling. The combination of Glove Results, F1 Micro starts from the stack of results from all classifiers in the order of GraphSAGE1, GraphSAGE2, FastGCN1, FastGCN1d, FastGCN2, FastGCN2d, SVM-poly, SVM-rbf, random forest and ADDA. Then they are combined across different source-target data pairs "CF"-"CO"-"OF"-"OC"-"FO"-"FC". The actual data used in Section 6.5 is the division of this data by classifier: DAGCN results and baseline results are separated. The data in the rest 7 rows are obtained and used in a similar matter.

33rd to 44th rows of Table E.1 are used in Section 6.4. Row 33rd to 40th are for comparison between different orders of normal oversampling. Not all permutations of embedding and source-target pairs are included. The exact cases are listed in Appendix C. Note that there is no Glove C-F Normal 80P and Glove C-O Normal 80P in 3rd order normal oversampling data. Row 41st to 44th are for comparison between normal oversampling and other oversampling methods. The exact cases included are listed in Appendix D.

In the table above, results of DAGCN using GraphSAGE classifier will be referred to as "graphsage1" or "graphsage2" depending on the graph construction method. Similarly, results from FastGCN classifier are labeled "fastgcn1" or "fastgcn2". Results from FastGCN classifier with discriminator are recorded as "fastgcn1d" or "fastgcn2d".

6.2 Comparison Between Two Graph Construction Methods

This section answers the second research question about graph construction method. Usually, scatter plot could give a broad overview on the distribution of the data. But when the amount of data is large and the difference between sub-categories of data is not very significant, scatter plot could not lead to a clear conclusion.

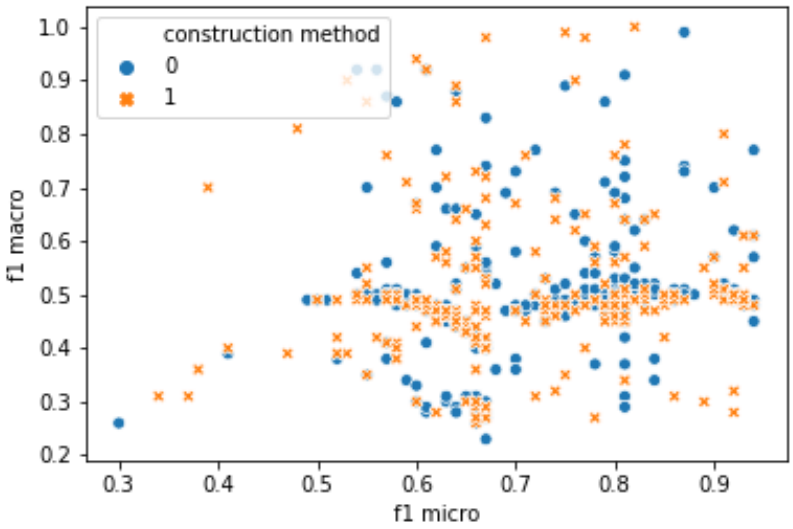

Figure 6.3: Scatter Plot: Construction Method Comparison, Full Data

Figure 6.3 are all the data for graph construction method comparison. Each data point's abscissa represents its f1 micro score and its ordinate represents f1 macro score. In total there are 518 data points. So it is difficult to observe a difference. Even if we split the data by the source-target datasets pairs to reduce the number of points, the trend of distribution could still be unclear, as Figure 6.4 shows.
Figure 6.4 only has the data acquired when the source dataset is CBASU and target dataset is Formspring. One may perceive that there is no systematical difference between these two class of result, but statistic test is still needed to make a certain conclusion.

We will compare all the f1 micro scores acquired through graph construction method 1 and 2. F1 micro score is equivalent to accuracy in our binary classification case. it is more important than f1 macro score, which only helps us understand class balance issue. Then we take the f1 macro score into consideration. The two dimensional result data (f1 micro, f1 macro) will be displayed through using scatter plot. Since the best situation for f1 macro score is not equal to 1, but equal to the corresponding f1 micro score, we take the absolute value of the difference between f1 micro score and f1 macro score, $D = |f1\ micro - f1\ macro|$. Now the two dimensional data become (f1 micro, D) and we can perform sign test on the dimension D. We name the f1 micro scores from construction method 1 $C1$ and the difference between f1 micro and macro for construction method 1 $D1$(name for construction method 2 is of a similar manner). The intermediate f1 micro result for sign test is calculated by $C1 - C2$ and the difference for sign test is $D1 - D2$. So if the null hypothesis is rejected, we will know which group of data tends to be larger based on the sign of test statistics.
As the result Table 6.2 shows, in terms of accuracy, there is no difference between these two graph

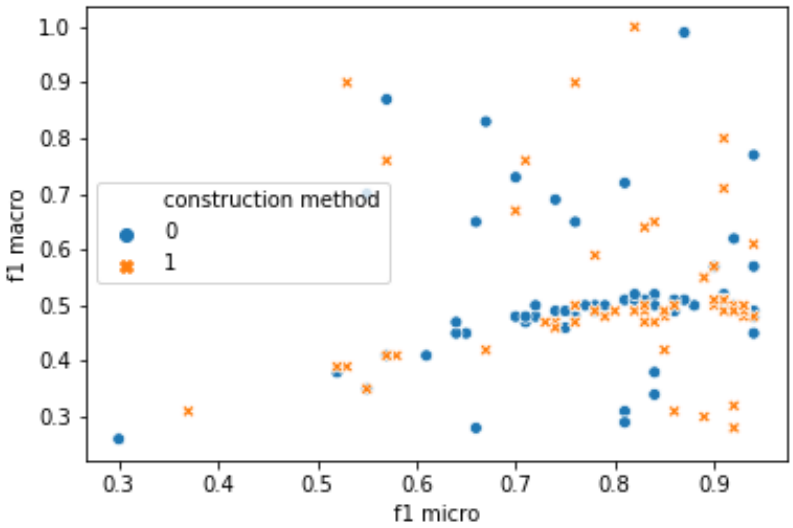

Figure 6.4: Scatter Plot: Construction Method Comparison, C-F

Data	**F1 Micro**	**F1 Micro and Macro Differences**
Statistics	-13.0	-26.0
p Value	0.09462	0.0009549

Table 6.2: Construction Method Comparison: All Data

construction method, as p value 0.095 is larger than 0.05. For the difference between f1 micro and f1 macro, the sign test result tells us that there exists a significant distinction between the two class of data, as the p value 0.0009550 is much smaller than 0.05. The test statistics -26.0 is negative, which means the difference for graph construction method 2 is larger. Class balance requires that difference to be as smaller as possible. Hence, we can conclude that the two graph construction method does not induce difference in accuracy, but has influence on class balance. Graph construction 1 has a more balanced performance.

6.3 Comparison Between DAGCN and Baselines

This section aims to answer the main research question—whether DAGCN methods have better performance than the baseline methods. In total, there are four baseline methods and six DAGCN variations. Ideally, a one-to-one comparison is succinct and easy to understand. But that would lead to too many pairs of comparisons. These will raise the p value greatly after Holm-Bonferroni correction. The following sessions are the comparison within baseline methods and DAGCN variations. After finding out the best performance among the two subgroups, there will be a one-to-one comparison to conclude if DAGCN could outperform baseline methods. In addition, selected DAGCN methods will be compared to ADDA to see if DAGCN is the better domain adaptation framework. In this section, only the no-oversampling cases are involved in the comparison. Because oversampling could only relieve the class imbalance problem at the expense of accuracy, it is not recommended in this project(we will come to that conclusion in later section). Hence, only the comparison result between no-oversampling method is needed. The oversampling cases are left out lest their greater number affects the conclusion.

6.3.1 Comparison Among Baseline Methods

First we will present some best results achieved by the four baseline methods individually as example. Then there will be the sign test comparison results. The signs for performing sign tests are acquired by let the results of the left side in "Methods for Comparison" in the tables minus the right side. Since only non-oversampling results are involved in comparison, the best results achieved by baseline methods when using oversampling are not listed here.

Methods	Embedding	Data Pairs	F1 Micro	F1 Macro
SVM,poly	W2V	C-F	0.94	0.68
	Glove	O-F	0.94	0.60
SVM,rbf	Glove	C-F	0.94	0.69
	W2V	C-F	0.94	0.68
Random Forest	W2V	C-F	0.94	0.65
	Glove	O-F	0.94	0.63
ADDA	W2V	O-F	0.86	0.49
	Bert	O-F	0.83	0.48

Table 6.3: Best Baseline Results Demonstration

The information in Table 6.3 shows that the best results from SVM-poly, SVM-rbf and Random Forest are at the same level, but the best results from ADDA are conspicuously below this level. This finding shows same inclination as the sign test comparisons, which are presented in Table 6.4 and Table 6.5.

Methods for Comparison	Statistics	p Value	p Value(HB)
SVM,poly vs. SVM,rbf	1.0	0.6875	0.6875
SVM,poly vs. Random Forest	-3.0	0.2379	0.4757
SVM,poly vs. ADDA	5.0	0.02128	0.06469
SVM,rbf vs. Random Forest	-5.0	0.02127	0.08508
SVM,rbf vs. ADDA	4.5	0.02246	0.08508
Random Forest vs. ADDA	8.0	0.0004025	0.002415

Table 6.4: Comparison of Baseline Methods: F1 Micro Score

Methods for Comparison	Statistics	p Value	p Value(HB)
SVM,poly vs. SVM,rbf	-1.5	0.5078	1.0
SVM,poly vs. Random Forest	1.0	0.8238	1.0
SVM,poly vs. ADDA	-5.5	0.007385	0.03692
SVM,rbf vs. Random Forest	0	1.0	1.0
SVM,rbf vs. ADDA	-4.5	0.02246	0.08984
Random Forest vs. ADDA	-10.0	1.907×10^{-6}	1.144×10^{-5}

Table 6.5: Comparison of Baseline Methods: Difference Between F1 Micro and F1 Macro

Table 6.4 suggests that Random Forest results have higher accuracy than ADDA. Table 6.5 shows that in terms of class balance, SVM-poly and Random Forest have better performance than ADDA; and there is no definitive superiority in other comparison pairs. Based on these information, we can conclude that Random Forest is the best baseline method, as it is the only one that dominates another in terms of accuracy and is of same level in class balance aspect. Therefore, Random Forest is taken as the best representative in baseline methods.

6.3.2 Comparison Among DAGCN Variations

The approach of comparison on DAGCN methods is very similar to that on baseline methods. First there will be best results demonstration for each of the DAGCN variations. Again results when using oversampling are not included.

Methods	Embedding	Data Pairs	F1 Micro	F1 Macro
GraphSAGE1	Glove	C-F	0.93	0.49
	Bert	O-F	0.93	0.48
GraphSAGE2	Bert	F-C	0.93	0.61
	ST	O-F	0.93	0.49
FastGCN1	W2V	C-F	0.94	0.77
	ST	O-F	0.94	0.61
FastGCN1d	Glove	C-F	0.94	0.49
	Bert	C-F	0.94	0.49
FastGCN2	ST	C-F	0.94	0.61
	ST	O-F	0.93	0.55
FastGCN2d	ST	C-F	0.94	0.48
	Bert	C-F	0.93	0.49

Table 6.6: Best DAGCN Results Demonstration

The results in Table 6.6 indicates that the accuracy of these DAGCN variations are very close, while class balance could be distinctive. FastGCN1 obviously has better performance on the class balance aspect.

Since Section 6.2 already concluded that graph construction method 1 is superior to method 2, we only consider the results from graphsage1, fastgcn1 and fastgcn1d.

Methods for Comparison	Statistics	p Value
graphsage1 vs. fastgcn1	-3.0	0.2631
graphsage1 vs. fastgcn1d	1.0	0.8318
fastgcn1 vs. fastgcn1d	0.5	1.0

Table 6.7: Comparison of DAGCN Methods: F1 Micro Score

Methods for Comparison	Statistics	p Value	p Value(HB)
graphsage1 vs. fastgcn1	4.0	0.2631	0.3031
graphsage1 vs. fastgcn1d	-5.5	0.03469	0.1040
fastgcn1 vs. fastgcn1d	-3.0	0.3074	0.3074

Table 6.8: Comparison of DAGCN Methods: Difference Between F1 Micro and F1 Macro

Through Table 6.7 and Table 6.8, we found that no p value is less than 0.05. Therefore no Holm-Bonferroni method is needed. We can directly conclude that these three method have no performance difference and use all of them to compare with Random Forest classifier.

6.3.3 Comparison Between DAGCN and Baseline

Here there will be comparison between DAGCN and Random Forest, as Random Forest is the best baseline method; and between DAGCN and ADDA, as ADDA is also a domain adaptation framework.

Methods for Comparison	Type of Result	Statistics	p Value
graphsage1 vs. Random Forest	F1 Micro	-1.5	0.6636
fastgcn1 vs. Random Forest	F1 Micro	1.5	0.8036
fastgcn1d vs. Random Forest	F1 Micro	-2.5	0.3833
graphsage1 vs. Random Forest	F1 Macro	0.5	1.0
fastgcn1 vs. Random Forest	F1 Macro	-2.0	0.5412
fastgcn1d vs. Random Forest	F1 Macro	3.0	0.2862

Table 6.9: Comparison Between DAGCNs and Baseline

All p values in Table 6.9 are larger than 0.05. Hence, we found no evidence that DAGCN methods could achieve better performance than the random forest classifier.

Methods for Comparison	Type of Result	Statistics	p Value	p Value(HB)
graphsage1 vs. ADDA	F1 Micro	11.0	2.980×10^{-6}	8.940×10^{-6}
fastgcn1 vs. ADDA	F1 Micro	10.0	3.588×10^{-5}	7.176×10^{-5}
fastgcn1d vs. ADDA	F1 Micro	8.0	4.024×10^{-4}	4.024×10^{-4}
graphsage1 vs. ADDA	F1 Macro	-9.5	6.604×10^{-5}	1.320×10^{-4}
fastgcn1 vs. ADDA	F1 Macro	-10.0	3.588×10^{-5}	1.076×10^{-4}
fastgcn1d vs. ADDA	F1 Macro	-8.0	4.024×10^{-4}	4.024×10^{-4}

Table 6.10: Comparison Between DAGCNs and ADDA

Table 6.10 is the comparison result between DAGCN and domain adaptation baseline ADDA. Now it is clearly showed that all p values after Holm-Bonferroni correction are lower than 0.05. And in the f1 micro group all tests have statistics larger than 0. In the f1 macro groups all tests have statistics smaller than 0. Therefore, these selected DAGCN methods have superior performance than ADDA in terms of accuracy and class balance.

6.4 Comparison Regarding Oversamplings

In this section, two comparison results will be introduced: comparison between results with no oversampling and normal oversampling, as well as between normal oversampling and ADASYN oversampling/maGAN oversampling. The first comparison aims to answer the third research question: Is oversampling the minority class(the bully class)helping alleviate the class imbalance problem?

6.4.1 Comparison of Oversamplings and No Oversamplings

The comparison of no sampling cases and normal sampling cases is divided into three stages: comparison of no sampling and 1st order oversampling;comparison of 1st order oversampling and 2nd order oversampling; comparison of 2nd order oversampling and 3rd order oversampling. Note that 1st order oversampling includes CBASU40P, Formspring 10P and OLID60P, 2nd order oversampling includes CBASU60P, Formspring 20P and OLID80P, 3rd order oversampling includes CBASU80P, Formspring 30P and OLID100P and so on. In each stage, sign tests are performed on the results produced by DAGCN variations(classification results from GraphSAGE, FastGCN and FastGCN with discriminator) and baseline results. Since the result used for comparison has two dimension– micro score and absolute difference between f1 micro and f1 macro, in total there

are 4 sign tests results.

The cases included in no sampling, 1st order oversampling and 2nd order oversampling are listed in Appendix C. All these results are concatenated to form a 1 dimensional array to perform one time sign test. The size of the array is 64 for baseline results and 96 for results from DAGCN methods. For 3rd order oversampling, The size of the array is 56 for baseline results and 84 for results from DAGCN methods(It does not contain Glove C-F and Glove C-O cases). When comparing 2nd order normal oversampling and 3rd order normal oversampling, these instances will be deleted from 2nd order normal oversampling group.

Like the comparison on baseline and DAGCN methods, in this section first there will be demonstration of best results for each oversampling order and method, then the sign test result analysis.

Oversampling	**Embedding**	**Data Pairs**	**Methods**	**F1 Micro**	**F1 Macro**
No oversampling	W2V	C-F	FastGCN1	0.94	0.77
	Glove	C-F	SVM,rbf	0.94	0.69
1st oversampling	W2V	O-F Normal60P	FastGCN1	0.94	0.69
	Glove	C-F Normal40P	Random Forest	0.94	0.68
2nd oversampling	Glove	C-F Normal60P	GraphSAGE2	0.93	0.49
	Glove	C-F Normal60P	Random Forest	0.92	0.66
3rd oversampling	W2V	C-F Normal80P	GraphSAGE1	0.92	0.50
	Bert	C-F Normal80P	SVM,poly	0.90	0.53
ADASYN	Glove	C-F ADASYN60P	ADDA	0.94	0.48
	Glove	C-F ADASYN40P	Random Forest	0.92	0.65
maGAN	Glove	C-F maGAN40P	SVM,rbf	0.94	0.69
	Glove	C-F maGAN60P	SVM,rbf	0.94	0.69

Table 6.11: Best Oversampling Results Demonstration

We can see from Table 6.11 that for normal oversamplings, both of the f1 micro and f1 macro scores will drop with the increasing of oversampling numbers. This trend is also partially verified with the sign test comparison, which will be introduced later. The permutation of data pairs for ADASYN and maGAN oversampling are much less than normal oversampling. That is probably the reason why their best results are all from baseline methods.

Table 6.12 shows a clear trend of oversampling: increasing the number of minority class leads to the drop of f1 micro score(Note that "No sampling" refers to "No Oversampling" and so on). All the p values of DAGCN results are smaller than the 0.05 threshold. Additionally, those three test statistics are all positive, which means the groups on the left side(No oversampling,1st oversampling and then 2nd oversampling) are consistently larger than the groups on the right side(1st oversampling, 2nd oversampling,and then 3rd oversampling). Hence, the conclusion here is that for DAGCN methods, oversampling causes accuracy to drop and the more minority instances are oversampled, the more accuracy will drop. But there is no such trend for baseline methods.

In Table 6.13, for baseline methods, there is a unwavering trend that the difference between f1 micro and f1 macro decreases as the oversampling of minority class increases. For DAGCN methods, this trend stops at the 2nd order oversampling. The orders of comparison groups are the same as Table 6.12. Top 5 p values are less than 0.05 threshold, which indicates the gap between f1 micro and f1 macro will definitely shrink for 1st and 2nd order oversampling. After 2nd order oversampling, the effect of reduce class balance will not work for DAGCN based classifiers, but

Comparison Groups	**Classifiers**	**Statistics**	**p Value**	**p Value(HB)**
No sampling vs. 1st sampling	Baseline	-1.0	0.8555	0.8555
No sampling vs. 1st sampling	DAGCN	17.5	0.0001266	2.532×10^{-4}
1st sampling vs. 2nd sampling	Baseline	4.5	0.2110	0.4220
1st sampling vs. 2nd sampling	DAGCN	13.0	0.006673	6.674×10^{-4}
2nd sampling vs. 3rd sampling	Baseline	6.0	0.04277	0.1283
2nd sampling vs. 3rd sampling	DAGCN	19.0	5.853×10^{-6}	1.755×10^{-5}

Table 6.12: The Trend of Oversampling: F1 Micro Score

Comparison Groups	**Classifiers**	**Statistics**	**p Value**	**p Value(HB)**
No sampling vs. 1st sampling	Baseline	11.0	1.951×10^{-4}	1.951×10^{-4}
No sampling vs. 1st sampling	DAGCN	16.0	0.001111	0.003332
1st sampling vs. 2nd sampling	Baseline	16.0	1.831×10^{-6}	5.494×10^{-6}
1st sampling vs. 2nd sampling	DAGCN	13.0	0.0066739	0.04116
2nd sampling vs. 3rd sampling	Baseline	13.5	1.485×10^{-5}	7.427×10^{-6}
2nd sampling vs. 3rd sampling	DAGCN	6.0	0.2007	0.2006

Table 6.13: The Trend of Oversampling: Difference Between F1 Micro and F1 Macro

only effective on baselines(SVMs, Random Forest and ADDA). Still, it can be concluded that oversampling reduces difference between f1 micro and f1 macro scores. But since it is already known that for DAGCN results, f1 micro score drops when the oversampling increases, additional tests need to be done to see if f1 macro scores are actually increasing.

Comparison Groups	**Classifiers**	**Statistics**	**p Value**	**p Value(HB)**
No sampling vs. 1st sampling	DAGCN	-12.5	0.009673	0.02902
1st sampling vs. 2nd sampling	DAGCN	-1.5	0.8264	0.9140
2nd sampling vs. 3rd sampling	DAGCN	-3.5	0.4570	0.9140

Table 6.14: The Trend of Oversampling: F1 Macro Score

Table 6.14 shows that only when comparing between no sampling cases and 1st oversampling cases from DAGCN results, does the f1 macro score rises with a statistical significance. This res-

ult suggests that oversampling methods work much more effectively for the baselines than for the DAGCNs. Although the experiment results shows that oversampling is good for baseline methods, for DAGCNs, it is not recommended to handle the class imbalance problem with oversampling the minority class. Because f1 macro score is only improved with 1st order oversampling, and oversampling is at a consistent cost of lowering accuracy, which is a more important indicator than the class balance.

6.4.2 Comparison of Oversamplings Methods

This section contains the comparison between normal oversampling and ADASYN oversampling, as well as the comparison between normal oversampling and maGAN oversampling. The instances included for comparison are listed in Appendix D. The number of instances is significantly smaller than normal oversampling, as the total instances in the other two oversampling approaches are small. Only the corresponding cases in normal oversampling will be used to compare with ADASYN or maGAN. The null hypotheses here for the sign tests are that the f1 micro scores(or difference between f1 micro and f1 macro) from normal oversampling are the same as that from ADASYN oversampling(or maGAN oversampling).

Comparison Groups	**Classifiers**	**Statistics**	p **Value**
Normal vs. ADASYN	Baseline	1.0	0.8450
Normal vs. ADASYN	DAGCN	4.5	0.2110
Normal vs. maGAN	Baseline	1.5	0.5488
Normal vs. maGAN	DAGCN	4.0	0.1338

Table 6.15: Comparison of Oversampling Methods: F1 Micro Score

Comparison Groups	**Classifiers**	**Statistics**	**p Value**
Normal vs. ADASYN	Baseline	-2.0	0.5846
Normal vs. ADASYN	DAGCN	6.0	0.1114
Normal vs. maGAN	Baseline	-2.0	0.4239
Normal vs. maGAN	DAGCN	5.0	0.05247

Table 6.16: Comparison of Oversampling Methods:Difference Between F1 Micro and F1 Macro

Table 6.15 and Table 6.16 shows that all the p values are larger than the threshold 0.05, hence there is no need to use Holm-Bonferroni correction method.And we can conclude directly that ADASYN and maGAN oversampling have no performance difference compared to normal oversampling.

6.5 Comparison Between Word Embedding Methods

We want to find out which word embedding method achieves the best performance for our project. All the f1 micro and macro scores of DAGCN acquired in different source-target datasets pairs are concatenated and compared in the same way in Section 6.3 to get the intended result. There is also a similar comparison for all the baseline results. As the oversampling cases are not equally distributed across all variations of word embeddings and source-target datasets pairs, here only the results from non-oversampling cases are used. Otherwise these unbalanced oversampling may cause the analysis result to be biased towards certain word embedding or source-target pairs scenarios.

Word Embedding	**G→ W**	**G→ B**	**W→ST**	**B→ ST**
Statistics, Baseline	-2.5	1.5	6.0	1.0
p Value,Baseline	0.3017	0.6290	0.001831	0.7744
p Value(HB), Baseline	0.9051	1.0	0.007324	1.0
Statistics, DAGCN	-9.0	-10.0	-0.5	3.0
p Value, DAGCN	5.335×10^{-4}	3.249×10^{-4}	1.0	0.3449
p Value(HB), DAGCN	0.001067	0.0009747	0.1655	0.3449

Table 6.17: Word Embedding Comparison: F1 Micro Score

Word Embedding	**G→ W**	**G→ B**	**W→ ST**	**B→ ST**
Statistics, Baseline	2.0	-0.5	-5.0	-3.0
p Value,Baseline	0.5234	1.0	0.04138	0.1795
p Value(HB), Baseline	1.0	1.0	1.0	0.5385
Statistics, DAGCN	-2.0	0.5	-3.5	-8.5
p Value, DAGCN	0.6075	1.0	0.3105	0.004551
p Value(HB), DAGCN	1.0	1.0	0.9315	0.01365

Table 6.18: Word Embedding Comparison: Difference Between F1 Micro and F1 Macro

From Table 6.17, we know that both W2V(Word2Vec) and Bert(B) has a better performance than for DAGCN methods, as their corrected p values are less than 0.05 and test statistics are negative. For baseline methods, it can be concluded that W2V(Word2Vec) has a better performance than ST(SentenceTransformer). Table 6.18 indicates that in terms of class balance, Bert(B) has a better result than SentenceTransformer(ST). (p value $0.01265 < 0.05$ for DAGCN results). Both W2V(Word2Vec) and ST(SentenceTransformer) surpass other embedding method, while not surpassed by another method. Word2Vec is more efficient in running times but it captures less information of the text,as its vocabulary is smaller than others and does not possess punctuations. In the future work, we can leave out Bert and Glove word embedding method to reduce the number of experiments.

6.6 Comparison About the Data Pairs' Origins

At Section 1.3, the 4th research question is about the assumption that although the data set CBASU and OLID are both from Twitter, the time discrepancy of their collections distances these two data set so that their relationship is similar to data sets from two platforms. This section will verify this assumption by comparing the non domain adaptation results of data pairs being CBASU and OLID(regardless of being source or target) to other results. Specifically, the comparisons are as follows:

1. CO+OC vs. CF+FC
2. CO+OC vs. FO+OC

It means the using concatenation of source-target pairs being CBASU-OLID and OLID-CBASU to compare with source-target pairs being CBASU-Formspring and Formspring-CBASU. For the second comparison it is a similar manner—"CO+OF" means Formspring-OLID and OLID-CBASU. The approach to combine data is shown in Figure 6.5:

In total there are 3 different classifiers, 4 different embedding and 2 different source-target combinations. So the length of data array for sign test is 48. Results from oversampling experiments are not included. If the results from data pairs being CBASU and OLID are better than data pairs being CBASU and Formspring, and also better than Formspring and OLID, then one explanation

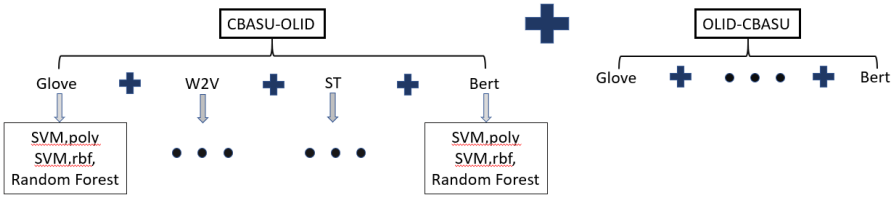

Figure 6.5: Illustration of Data Combination

could be that they are actually from the same domain. The sign test results are placed in Table 6.19:

Comparison	**Data Type**	**Statistics**	**p Value**	**p Value(HB)**
CO+OC vs. FO+OF	F1 Micro	-1.0	0.7905	0.7905
CO+OC vs. FO+OF	Micro-macro Difference	-7.5	6.103×10^{-5}	1.221×10^{-4}
CO+OC vs. CF+FC	F1 Micro	-4.5	0.04904	0.09567
CO+OC vs. CF+FC	Micro-macro Difference	-6.5	0.002349	0.002349

Table 6.19: Comparison About the Data Pairs' Origins

The table above shows that in terms of accuracy, there are no difference between the results from both Twitter data sets and other combinations. However, in the class balance perspective, Twitter data sets as source-target do have better results. In this case, the assumption that the two Twitter data set can be treated as from different platform cannot be verified and need more corroborations.

6.7 Individual Cases Analysis

This section presents some well selected instances from the Formspring dataset, which is subsequently converted to Word2Vec embedding, and predicted by FastGCN classifier. There will be examples that are correctly classified as bully, correctly classified as non-bully, falsely classified as bully and falsely classified as non-bully. The follow-up discussion includes the content of the instance itself and the reasonableness of original label. Note that in the tables below, being labeled 1 means it is a bully instance, and being labeled 0 means non-bully.

Table 6.20 provides 3 examples of non-bully data that are correctly classified as non-bully. The first example is an ordinary conversation that contains no component which could be interpreted as aggressive, bullying, dirty or filthy. It is very reasonable to get this prediction result. The second example contains the word "bitch". But both the annotator and our classifier recognize that this is an intimate address between friends. In the third example, there is again an expression "stupid bitches" that might be construed as bully. In this case however, "stupid bitches" is not target to any specific person or group of people. Hence it is not counted as bully, as both the annotator and our classifier do. This proves that the DAGCN classifier is able to distinguish at least some non-bully instances even though they contain component that is usually treated as bullying.

Order Number	Text Content	Original Label	Prediction
6th	Q: Are you a morning or night person?A: Night 4shuree!!	0	0
41st	Q: Hello bitch! How's ur day going? Good I hope :) it's beautiful and sunny! A: Ohh heyy bitchh!!! its goin good..:D thx 4 askin	0	0
66th	Q: If you could rid the world of one thing what would it be? A: HOEISHNESS! lol um prob. stupid bitches and SIN! i just sinned!Opppzz	0	0

Table 6.20: Original Textual Examples of Cyberbully Data,Correct Non-bully

Order Number	Text Content	Original Label	Prediction
34th	Q: Hahah. Funny how u defend that beiber kid. A: its also funny how u stalked my whole twitter! Nice goin! BITCH!	1	1
213rd	Q: Chick? Are you a fucking lesbain?A: No are you	1	1
236th	Q: dude it seems like everyone hates you. you should just delete all your socialing sites. (:A: should i care?	1	1

Table 6.21: Original Textual Examples of Cyberbully Data,Correct Bully

Table 6.21 shows the examples of correctly predicted bully instances. The second example is a typical bullying instance that targets not only the person who gives the answer also the lesbian people. The first example is a controversial bully instance. The person who answer the question said "bitch" but it is a retaliation for the questioner's cyberstalking behavior. If the cyberstalking behavior does exist then it becomes an acceptable response. For the third example, it contains no words or phrases that might be linked to bullying. But on the whole, the meaning of the question is a personal attack to the answerer. Almost any person receiving this kind of message from a stranger could be annoyed, irritated or frustrated. So this instance should be treated as having bullying content. And it proves that our classifier is capable of recognizing bullying meaning even if the instance has no explicit bullying words or phrases.

Table 6.22 gives some examples of DAGCN classifier falsely predicts bully instances as non-bully. The first example is at the boundary between bully and non-bully. It does contain bully component in the question, but the relationship between the two people and the real attitude of the answerer is not known to us. It could be teasing of friends or bullying from a stranger. That is one major shortage for this cyberbully detection branch: taking the "repeat" component off the definition also large reduces the background information. The second example is obviously

Order Number	Text Content	Original Label	Prediction
47th	Q: Hey. Why you such a bitch? A: Why thank yuh!	1	0
51st	Q: how many languages do u speak?A: Umm.. i think 2 but english is my first! im learnin french! d u?!	1	0
197th	Q: :3 wow ashleeee your fakes amuse me lmao okay you are retarded to keep this page upp :] A: your retarded	1	0

Table 6.22: Original Textual Examples of Cyberbully Data,False Non-bully

an error made by the annotators. It definitely does not contain bullying component, but labeled bully. And our classifier correctly recognized it as non-bully. The mis-classification in the third example highlights several shortages of the DAGCN classifier. The whole classification process is total unrelated to the semantics of the instances. It is not explainable. And averaging the word embeddings to form a sentence embedding might lead to unpredictable result-such as making the average of words that has bullying meaning very close to an embedding of non-bully sentence.

Order Number	Text Content	Original Label	Prediction
25th	Q: Do yuh have any stalkers? A: Hell yes! Lots of them and thats including yuh!<3	0	1
32nd	Q: haha check inside ur closet u mind find out who it is(; A: Lmao!! kk lemme get my ass off this toliet seat then ill check!! Mwahah!!	0	1
56th	Q: If you could ask Barack Obama one question what would it be? A: Oh snap!! I would actually hug him first off ahah and say Nigga where my weed go? Hahaha jkjkjk i love him!! He is doin a great job!	0	1

Table 6.23: Original Textual Examples of Cyberbully Data,False Bully

Table 6.23 gives 3 examples of DAGCN classifier incorrectly predicts non-bully instances as bully. The possible explanation for the mistake in the first example is that it contains element that shows

strong, even aggressive feeling(the use of word "hell") and it is toward the questioner. The reason for incorrect prediction of the second and the third example is the same: they contain words that itself might be understand as bully("ass" in the second and "Nigga" in the third) but the meaning of whole sentence is not.

The discussions on the listed examples above shows that DAGCN is better than the simple classifier based on restricted word lists, as it can grasp the bully sentence without explicit bullying related words and recognize the non-bully meaning sentence with bullying related words. However, it is not alway capable of doing that. And sometime it will miss some very obvious bullying cases or incorrectly predict non-bully instance as bully.

Chapter 7

Conclusion

This chapter will summarize the research questions brought up in Section 1.3 and the analysis in Chapter 6, and draw conclusions towards the 4 research questions. Then we will discuss the some of the limitations of the current work, and the planned improvements for the future.

7.1 Main Contributions

The main research question we investigated in this thesis is whether the proposed DAGCN method out-performs the state-of-the-art methods for cyberbully detection. Unfortunately, through the test results in Section 6.3, there is no evidence that the complicated DAGCN achieves better performance than the random forest classifier, which has the best performance among the baseline methods. But all DAGCN variations cost much more time and memory space than the random forest classifier. Running GraphSAGE classifier alone cost 30-40 minutes. FastGCN classifier is indeed much faster, usually between 5 minutes to 15 minutes. In addition, running the graph construction codes cost 30 minutes. It can be argued that once optimized, these codes' running time will be significantly reduced. But random forest classifier cost 5-10 minutes, which is lower. And it requires much less memory and CPU resources. Hence, the conclusion for this question is definite: there is no proof that DAGCN method could have better performance than the simple random forest method.

The performance of DAGCN approach is also influenced by its graph construction method. This is raised in the second research question and later investigated in Section 6.2. The conclusion from sign test is that the two graph construction methods have equal performance in terms of accuracy, but method 1 is better in perspective of class balance. Although this project only experiment two graph construction method, a performance distinction already emerges between them. Considering the fact that these two graph construction method only differ in the number of bridge nodes and source-bridge edges per bridge node, it naturally comes to mind that more changes in the graph construction method may improves performance to a greater degree. Adopting a better graph construction could not only relieve the class imbalance issue, but increase the accuracy as well. The importance of graph construction in the realm of graph-based semi- supervised learning make it worthwhile further exploring in future works.

The third research question proposes that oversampling the minority class in the source data set could alleviate the class imbalance problem in the prediction result. Section 6.4.1 illustrates how oversampling of the minority class in the source data affects the prediction result. For baseline methods, it works very well—class imbalance is relieved while the accuracy is maintained. But for DAGCN method, the class imbalance is reduced at the cost of drop in accuracy, while f1 macro scores of DAGCN classifier results are only ameliorated for the first order of oversampling. So the conclusion is that oversampling minority class in source data is not suitable in this project.

F1 macro scores does increase with 1st order oversampling, together with the sign test result on f1 micro-macro difference, is a sign of successfully relieving class imbalance. However, f1 micro score is equivalent to accuracy, which bears heavier importance than f1 macro score. So there needs to be a metric judging whether the trade-off between these two f1 scores is appropriate. Due to the lack of such metric, this project cannot use oversampling to resolve the class imbalance problem. Developing such a metric for trade off between accuracy and class imbalance will be a very interesting work in the future. Regarding oversampling methods, this project finds that ADASYN, maGAN and normal oversampling methods achieve very similar results.

The last research question is about the assumption that the two Twitter data set(CBASU and OLID) can be treated as from different platform because their dates of collection are too far from each other. This assumption is analysed In Section 6.6. The results from source-target data pairs shows that CO and OC does not differ in accuracy from other two kinds(CF+FC and FO+OF) of source-target data pairs, they do show different performance in terms of class balance. This cast doubt on our previous assumption. In the future work, there should be more experiments to explore this assumption in more details.

This project also concludes that adding a discriminator for the FastGCN classifier to fill the possible distribution gap of intermediate embedding of bullying and non-bully instances does not achieve better result. In Section 6.3.2, the sign test result shows that FastGCN1 and FastGCN1d do not differ in accuracy and class balance. Moreover, with discriminator the FastGCN classifier cost more time. Therefore it is not recommended.

7.2 Limitations

7.2.1 Definition of Cyberbully Detection

The divergent definition on cyberbully detection has caused some extra effort at the beginning of this project. Two major branches of the definition differ on whether cyberbully should include the “repeat” feature. On the first glance they may seem similar, but if cyberbullying must be a repeat behavior, then much more information is needed. For one-time bullying, the researchers could only gather the text itself, as the CBASU data in this project. Identifying repeating bully not only increases the text needed, but the user and time information are required as well. Because the repeated bully has to be between these same two users and in a reasonable short time. For one-time bullying, the target of the cyberbully detection classifier is the text; for repeated bullying, the target becomes the relationship between these two users in a certain period, which is more complex. In addition to the text processing, a cyberbully detection classifier for repeated bullying should also consider the context of these text, such as if the users are strangers or friends.

The complication of repeating cyberbully detection problem leads to the complication of classifiers. The classifiers for repeating bullying could become too complex that they bear no meaning as reference for people with the intention to study one-time cyberbully detection problem. Hence, the mixed definition of cyberbully detection is bound to cause confusion and time waste for new researchers in this field. In the future, the development of this realm will certain widen the gap between these two generation of classifiers and growth of paper in number will cause more waste in time for researchers. Therefore, giving distinctive names for the two branches of cyberbully detection is very important. It is of the top priority in our future work.

7.2.2 Dataset Selection

Usually, the instances marked as ’bully” are much less in proportion to that in CBASU and OLID dataset in real world cyberbully detection scenario. [32] reported that around 6% of Tweets contain bullying information. The datasets CBASU and OLID are drawn from Twitter, but their

cyberbullying proportions are much larger: CBASU has around 25% of instances marked as bully and OLID has 33% bully instances. The ratio difference could lead to performance gap in the f1 macro scores. As the distribution of bully and non-bully instances are even more unbalanced, it ought be more difficult to maintain balanced result between two class for real world data.

7.2.3 Graph Construction

Despite the fact that this project does not possess much discussion or detailed experiment on graph construction, its method actually has a great influence in semi-supervised learning,according to [51] and [60]. The k nearest neighbors method we use in this project is bound to lead severe uneven edge degree distribution per node, and this weakens the classifier's performance. In addition to graph construction method, the whole graph construction time is too long at the current stage(around 30 minutes). And one of the most time-consuming part for kNN graph construction is to search for nearest neighbors for each node. Optimizing the graph construction process will allow us to run more experiments.

7.2.4 Baseline

The high proportion of invalid results in ADDA(19 invalid results out of 48) is one limitation on the conclusion that DAGCNs have better performance than the baseline domain adaptation framework ADDA. This unfavorable result from ADDA leads to the suspicion that the reason that other methods outperforms ADDA is that the ADDA setting in this project is unsuitable for this specific cyberbully detection problem. As a popular domain adaptation framework, ADDA ought to yield better results.

7.2.5 Statistic Test

This project uses sign test for analysis as it has the least requirements on data distributions. However, the statistic power of sign test is also among the weakest statistic tests. This might cause the test result unable to distinguish the performance differences between certain methods. A stronger statistic test could yield a more distinctive result.

7.3 Future Work

In accordance with the limitations raised in Section 7.1 and Section 7.2.1, the future work planned for this project includes:

1. **Definition of cyberbully detection:** The first item on the future work list is to change the name "cyberbully detection" in our project to other names, such as cyber-abuse. Also, we should urge other researchers doing similar research in the community to change the "cyberbully" name used in their projects too.

2. **Dataset Selection:** Try to build our own dataset according to our special goals. Many public dataset do not introduces how they are parsed and processed, and we want a dataset that mimics the real platform situation. Our dataset could be collected from a specific topic duration a specific short time period(could be when the topic is popular) under some platform.

3. **Graph Construction:** In the future work, DAGCN could be tested through the b-matching graph construction method proposed by [25]. B-matching not only could make the graph symmetric(meaning the two nodes connected by an edge are both in each other's k neighborhood), but it also even the edge degree for the nodes as well. We could expect that the improved graph construction method will increase the classifier's performance. For the long running time of graph construction, [26] proposed a more efficient similarity searching

method, FAISS, for high dimensional dense vectors. The future work will include FAISS to reduce the time and memory cost during the graph construction period.

4. **ADDA baseline:** There are several possibilities that ADDA produces so many invalid results, such as underfitting, overfitting or converging at sub-optimal result. Future work should include running more tests to determine which problem the current ADDA framework has, and to reduce the invalid results proportion to a same level as other baseline methods.

5. **Statistic Test:** Future works could try Box-Cox transformation on the data to make them conform to a normal distribution. Even if after transformation, the data are unable to pass Shapiro-Wilk test so that paired t test cannot be used, reducing the existing substantial non-normality of our data is still beneficial for statistic tests that do not demand normality[33]. Also, we should devise a better way to process invalid results, as the current way severely disrupts the data's original distribution. The most powerful test, paired t test, does not strictly requires normal distribution. But if the distribution of data is far away from normal distribution,then it will not be effective. We should try to make paired t test usable in the future work.

6. **F1 Micro-Macro Trade-off Metric:** Applying oversampling method to DAGCN method increases f1 macro score at the cost of reducing f1 micro score. Thus there should be a metric to judge how much f1 macro score increase is worth 1 percent reduction in f1 micro score, and to what extent this metric is suitable. Without this metric, all methods that causes f1 micro score to drop have to be rejected. Our current vague idea is that 3 percent increase of f1 macro score is tradable for 1 percent reduction of f1 micro score and the reduction of f1 micro score has to be within the range of 3 percent.

7. **Distribution Difference in the two Twitter Datasets:** Although the CBASU and OLID datasets are both from Twitter, this project assumes that they should be treated as if being from different social platforms because of the large gap on their collecting date. This assumption is questioned by the results in Section 6.6. In the future, we should analyse the results from domain adaptation classifiers, comparing the results' differences between data pairs being CBASU-OLID and other two cases(CBASU-Formspring and OLID-Formspring). If CBASU-OLID's performance is at the same level as the other two in accuracy and better in class imbalance, which is the same conclusion in Section 6.6, then it will still be reasonable to treat them as being from different domains.

Bibliography

[1] 3.2.4.3.1. sklearn.ensemble.randomforestclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Accessed: 2020-08-11. 18

[2] Boole's inequality. https://en.wikipedia.org/wiki/Boole%27s_inequality#Bonferroni_inequalities. Accessed: 2020-09-05. 23

[3] F1 score. https://en.wikipedia.org/wiki/F1_score. Accessed: 2020-09-09. v, 24

[4] Holm–bonferroni method. https://en.wikipedia.org/wiki/Holm-Bonferroni_method#Formulation. Accessed: 2020-09-05. 23

[5] imbalanced-learn. https://imbalanced-learn.readthedocs.io/en/stable/. Accessed: 2020-09-28. 19

[6] sklearn.svm.svc. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2020-08-11. 18

[7] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. 2001. 6

[8] Uwe Bretschneider, Thomas Wöhner, and Ralf Peters. Detecting online harassment in social networks. 2014. 4

[9] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018. v, 14, 15

[10] Lu Cheng, Jundong Li, Yasin Silva, Deborah Hall, and Huan Liu. Pi-bully: Personalized cyberbullying detection with peer influence. In *IJCAI*, 2019. 18

[11] Lu Cheng, Jundong Li, Yasin N Silva, Deborah L Hall, and Huan Liu. Xbully: Cyberbullying detection within a multi-modal context. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 339–347, 2019. 5

[12] Maral Dadvar and Franciska De Jong. Cyberbullying detection: a step toward a safer internet yard. In *Proceedings of the 21st International Conference on World Wide Web*, pages 121–126, 2012. 4

[13] Maral Dadvar, FMG de Jong, Roeland Ordelman, and Dolf Trieschnigg. Improved cyberbullying detection using gender information. In *Proceedings of the Twelfth Dutch-Belgian Information Retrieval Workshop (DIR 2012)*. University of Ghent, 2012. 4

[14] Maral Dadvar, Roeland Ordelman, Franciska de Jong, and Dolf Trieschnigg. Towards user modelling in the combat against cyberbullying. In *International Conference on Application of Natural Language to Information Systems*, pages 277–283. Springer, 2012. 4

[15] Maral Dadvar, Dolf Trieschnigg, and Franciska de Jong. Experts and machines against bullies: A hybrid approach to detect cyberbullies. In *Canadian Conference on Artificial Intelligence*, pages 275–281. Springer, 2014. 4

[16] Maral Dadvar, Dolf Trieschnigg, Roeland Ordelman, and Franciska de Jong. Improving cyberbullying detection with user context. In *European Conference on Information Retrieval*, pages 693–696. Springer, 2013. 4

[17] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th international conference on Machine learning*, pages 193–200, 2007. 5

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. v, 12, 13

[19] Karthik Dinakar, Roi Reichart, and Henry Lieberman. Modeling the detection of textual cyberbullying. In *fifth international AAAI conference on weblogs and social media*, 2011. 4

[20] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014. 5

[21] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017. v, 14

[22] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, pages 1322–1328. IEEE, 2008. 19

[23] Homa Hosseinmardi, Sabrina Arredondo Mattson, Rahat Rafiq, Richard Han, Qin Lv, and Shivakant Mishra. Poster: Detection of cyberbullying in a mobile social network: Systems issues. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 481–481, 2015. 4

[24] Qianjia Huang, Vivek Kumar Singh, and Pradeep Kumar Atrey. Cyber bullying detection using social and textual analysis. In *Proceedings of the 3rd International Workshop on Socially-Aware Multimedia*, pages 3–6, 2014. 4

[25] Tony Jebara, Jun Wang, and Shih-Fu Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 441–448, 2009. 6, 42

[26] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019. 42

[27] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 14

[28] Wouter M Kouw and Marco Loog. An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*, 2018. 5

[29] Xiaobo Liu, Zhentao Liu, Guangjun Wang, Zhihua Cai, and Harry Zhang. Ensemble transfer learning algorithm. *IEEE Access*, 6:2389–2396, 2017. 5

[30] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. v, 10, 11

[31] Vinita Nahar, Sayan Unankard, Xue Li, and Chaoyi Pang. Sentiment analysis for effective detection of cyber bullying. In *Asia-Pacific Web Conference*, pages 767–774. Springer, 2012. 4

[32] Charles E Notar, Sharon Padgett, and Jessica Roden. Cyberbullying: A review of the literature. *Universal Journal of Educational Research*, 1(1):1–9, 2013. 41

[33] Jason Osborne. Improving your data transformations: Applying the box-cox transformation. *Practical Assessment, Research, and Evaluation*, 15(1):12, 2010. 43

[34] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2010. 5

[35] Yulong Pei. Graph convolutional networks for unsupervised domain adaptation. preprint on webpage at math.rochester.edu/people/faculty/cohf. 2

[36] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. v, 12

[37] Rahat Ibn Rafiq, Homa Hosseinmardi, Richard Han, Qin Lv, Shivakant Mishra, and Sabrina Arredondo Mattson. Careful what you share in six seconds: Detecting cyberbullying instances in vine. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 617–622. IEEE, 2015. 4

[38] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011. 26

[39] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019. 13

[40] Kelly Reynolds, April Kontostathis, and Lynne Edwards. Using machine learning to detect cyberbullying. In *2011 10th International Conference on Machine learning and applications and workshops*, volume 2, pages 241–244. IEEE, 2011. 18

[41] Semiu Salawu, Yulan He, and Joanna Lumsden. Approaches to automated detection of cyberbullying: A survey. *IEEE Transactions on Affective Computing*, 2017. 4

[42] Allison M Schenk and William J Fremouw. Prevalence, psychological impact, and coping of cyberbully victims among college students. *Journal of school violence*, 11(1):21–37, 2012. 1

[43] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 13

[44] Stephen M Serra and Hein S Venter. Mobile cyber-bullying: A proposal for a pre-emptive approach to risk mitigation by employing digital forensic readiness. In *2011 Information Security for South Africa*, pages 1–5. IEEE, 2011. 4

[45] Vivek K Singh, Souvick Ghosh, and Christin Jose. Toward multimodal cyberbullying detection. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2090–2099, 2017. 5

[46] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. *arXiv preprint arXiv:1511.05547*, 2015. 15

[47] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer, 2016. 5

[48] Robert S Tokunaga. Following you home from school: A critical review and synthesis of research on cyberbullying victimization. *Computers in human behavior*, 26(3):277–287, 2010. 1

[49] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4068–4076, 2015. 5

[50] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7167–7176, 2017. 18

[51] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020. 6, 13, 42

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 12

[53] Chang Wan, Rong Pan, and Jiefei Li. Bi-weighting domain adaptation for cross-language text classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. 5

[54] Ruohan Wang, Antoine Cully, Hyung Jin Chang, and Yiannis Demiris. Magan: Margin adaptation for generative adversarial networks. *arXiv preprint arXiv:1704.03817*, 2017. 19

[55] Jun-Ming Xu, Kwang-Sung Jun, Xiaojin Zhu, and Amy Bellmore. Learning from bullying traces in social media. In *Proceedings of the 2012 conference of the North American chapter of the association for computational linguistics: Human language technologies*, pages 656–666. Association for Computational Linguistics, 2012. 4

[56] Yonghui Xu, Sinno Jialin Pan, Hui Xiong, Qingyao Wu, Ronghua Luo, Huaqing Min, and Hengjie Song. A unified framework for metric transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 29(6):1158–1171, 2017. 5

[57] Marcos Zampieri, Shervin Malmasi, Preslav Nakov, Sara Rosenthal, Noura Farra, and Ritesh Kumar. Predicting the Type and Target of Offensive Posts in Social Media. In *Proceedings of NAACL*, 2019. 18

[58] Rui Zhao, Anna Zhou, and Kezhi Mao. Automatic detection of cyberbullying on social networks based on bullying features. In *Proceedings of the 17th international conference on distributed computing and networking*, pages 1–6, 2016. 5

[59] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003. 6

[60] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005. 20, 42

[61] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002. 6

[62] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 11249–11259, 2019. 16

Appendix A

Full List of Oversampling Cases

Embedding Type	Source Data	Target Data	Oversampling Method
Glove	CBASU	OLID	maGAN40P
Glove	CBASU	OLID	maGAN60P
Glove	CBASU	OLID	maGAN80P
Glove	CBASU	OLID	ADASYN40P
Glove	CBASU	OLID	ADASYN60P
Glove	CBASU	OLID	Normal40P
Glove	CBASU	OLID	Normal60P
Glove	CBASU	Formspring	ADASYN40P
Glove	CBASU	Formspring	ADASYN60P
Glove	CBASU	Formspring	Normal40P
Glove	CBASU	Formspring	Normal60P
Glove	CBASU	Formspring	maGAN40P
Glove	CBASU	Formspring	maGAN60P
Glove	CBASU	Formspring	maGAN80P
Glove	Formspring	OLID	ADASYN10P
Glove	Formspring	OLID	ADASYN20P
Glove	Formspring	OLID	ADASYN30P
Glove	Formspring	OLID	ADASYN40P
Glove	Formspring	OLID	ADASYN50P
Glove	Formspring	OLID	Normal10P
Glove	Formspring	OLID	Normal20P
Glove	Formspring	OLID	Normal30P
Glove	Formspring	OLID	Normal40P
Glove	Formspring	OLID	Normal50P
W2V	OLID	CBASU	Normal60P
W2V	OLID	CBASU	Normal80P
W2V	OLID	CBASU	Normal100P
W2V	OLID	Formspring	Normal60P
W2V	OLID	Formspring	Normal80P
W2V	OLID	Formspring	Normal100P
W2V	Formspring	CBASU	Normal10P
W2V	Formspring	CBASU	Normal20P
W2V	Formspring	CBASU	Normal30P
W2V	Formspring	CBASU	Normal40P

W2V	CBASU	Formspring	Normal40P
W2V	CBASU	Formspring	Normal60P
W2V	CBASU	Formspring	Normal80P
ST	CBASU	Formspring	Normal40P
ST	CBASU	Formspring	Normal60P
ST	CBASU	Formspring	Normal80P
ST	CBASU	OLID	Normal40P
ST	CBASU	OLID	Normal60P
ST	CBASU	OLID	Normal80P
Bert	Formspring	OLID	Normal10P
Bert	Formspring	OLID	Normal20P
Bert	Formspring	OLID	Normal30P
Bert	Formspring	OLID	Normal40P
Bert	Formspring	CBASU	Normal10P
Bert	Formspring	CBASU	Normal20P
Bert	Formspring	CBASU	Normal30P
Bert	Formspring	CBASU	Normal40P
Bert	CBASU	Formspring	Normal40P
Bert	CBASU	Formspring	Normal60P
Bert	CBASU	Formspring	Normal80P
Bert	CBASU	Formspring	Normal100P
Bert	CBASU	OLID	Normal40P
Bert	CBASU	OLID	Normal60P
Bert	CBASU	OLID	Normal80P
Bert	CBASU	OLID	Normal100P
Bert	OLID	CBASU	Normal60P
Bert	OLID	CBASU	Normal80P
Bert	OLID	CBASU	Normal100P
Bert	OLID	Formspring	Normal60P
Bert	OLID	Formspring	Normal80P
Bert	OLID	Formspring	Normal100P

Table A.1: All Oversampling Cases

Oversampling	**Minority Number**	**Ratio of Oversampled Minority on Majority**
F10P	1193	1.543
F20P	2385	3.085
F30P	3578	4.629
F40P	4770	6.17
F50P	5963	7.71
C40P	6459	1.679
C60P	9688	2.519
C80P	12918	3.360
C100P	16148	4.2
O60P	5304	1.205
O80P	7072	1.607
O100P	8840	2.01

Table A.2: Minority Numbers in Oversampling

Appendix B

Source-bridge edges and Bridge number For the First Construction Method

Embedding	**Source-Target**	**Oversampling Method**	**Source-Bridge Edges Degree**	**Bridge Nodes /Target Nodes**
Glove	C-O	None	14	5337/7886
Glove	C-F	None	11	2095/10602
Glove	F-C	None	25	3333/16652
Glove	F-O	None	55	2198/11026
Glove	O-F	None	3	47/12651
Glove	O-C	None	3	57/19928
Glove	C-O	maGAN40P	10	1642/11582
Glove	C-O	maGAN60P	10	687/12537
Glove	C-O	maGAN80P	10	753/12471
Glove	C-O	ADASYN40P	11	2547/10677
Glove	C-O	ADASYN60P	14	2547/10677
Glove	C-O	Normal40P	14	2550/10674
Glove	C-O	Normal60P	11	2539/10685
Glove	C-F	ADASYN40P	11	2095/10602
Glove	C-F	ADASYN60P	11	2095/10602
Glove	C-F	Normal40P	11	2095/10602
Glove	C-F	Normal60P	11	2095/10602
Glove	C-F	maGAN40P	10	133/12562
Glove	C-F	maGAN60P	10	791/11907
Glove	C-F	maGAN80P	10	804/11894
Glove	F-O	ADASYN10P	53	2514/10710
Glove	F-O	ADASYN20P	53	2514/10710
Glove	F-O	ADASYN30P	53	2547/10677
Glove	F-O	ADASYN40P	53	2514/10710
Glove	F-O	ADASYN50P	53	2514/10710
Glove	F-O	Normal10P	53	2514/10710
Glove	F-O	Normal20P	53	2514/10710
Glove	F-O	Normal30P	53	2514/10710
Glove	F-O	Normal40P	53	2514/10710

Glove	F-O	Normal50P	53	2514/10710
W2V	C-O	None	55	2827/10388
W2V	C-F	None	40	3051/9583
W2V	F-C	None	15	3651/16059
W2V	F-O	None	15	2127/11088
W2V	O-F	None	20	2862/9772
W2V	O-C	None	20	3929/15781
W2V	C-F	Normal40P	40	3015/9619
W2V	C-F	Normal60P	40	3051/9583
W2V	C-F	Normal80P	40	3054/9580
W2V	C-O	Normal40P	55	2827/10388
W2V	C-O	Normal60P	55	2840/10375
W2V	C-O	Normal80P	55	2865/10350
W2V	O-C	Normal60P	20	3929/15781
W2V	O-C	Normal80P	20	3933/15777
W2V	O-C	Normal100P	20	3930/15780
W2V	O-F	Normal60P	20	2860/9774
W2V	O-F	Normal80P	20	2861/9773
W2V	O-F	Normal100P	20	2859/9775
W2V	F-C	Normal10P	20	2935/16775
W2V	F-C	Normal20P	20	2921/16789
W2V	F-C	Normal30P	20	2923/16787
W2V	F-C	Normal40P	20	2923/16787
ST	C-O	None	12	5119/8120
ST	C-O	Normal40P	13	944/12295
ST	C-O	Normal60P	13	944/12295
ST	C-O	Normal80P	12	5119/8120
ST	C-F	None	13	2829/9869
ST	C-F	Normal40P	13	2829/9869
ST	C-F	Normal60P	13	2829/9869
ST	C-F	Normal80P	13	2829/9869
ST	F-C	None	67	2498/17495
ST	F-O	None	60	3425/9815
ST	O-F	None	10	4274/8424
ST	O-C	None	13	3067/16926
Bert	C-F	None	18	2573/10125
Bert	C-F	Normal40P	18	2573/10125
Bert	C-F	Normal60P	18	2573/10125
Bert	C-F	Normal80P	18	2573/10125
Bert	C-F	Normal100P	18	2573/10125
Bert	C-O	None	12	2025/11215
Bert	C-O	Normal40P	12	2025/11215
Bert	C-O	Normal60P	12	2029/11211
Bert	C-O	Normal80P	12	2044/11196
Bert	C-O	Normal100P	12	2064/11176
Bert	O-F	None	8	2352/10345
Bert	O-F	Normal60P	8	2352/10345
Bert	O-F	Normal80P	8	2357/10340
Bert	O-F	Normal100P	8	2377/10320
Bert	O-C	None	8	3988/16005
Bert	O-C	Normal60P	10	2322/17671

Bert	O-C	Normal80P	8	2323/17670
Bert	O-C	Normal100P	8	2377/17616
Bert	F-O	None	32	2472/11215
Bert	F-O	Normal10P	35	2136/11104
Bert	F-O	Normal20P	35	2136/11104
Bert	F-O	Normal30P	35	2136/11104
Bert	F-O	Normal40P	35	2136/11104
Bert	F-C	None	30	2984/17009
Bert	F-C	Normal10P	35	2442/17511
Bert	F-C	Normal20P	35	2442/17511
Bert	F-C	Normal30P	35	2442/17511
Bert	F-C	Normal40P	35	2940/17053

Table B.1: Bridge Nodes Number and Source-Bridge Edges Degree

Appendix C

List of Cases Included for Comparison Between No Sampling, 1st Order Normal Oversampling and 2nd Order Normal Oversampling

Embedding	**Source**	**Target**
Glove	CBASU	Formspring
W2V	CBASU	Formspring
Glove	CBASU	OLID
W2V	CBASU	OLID
W2V	OLID	Formspring
W2V	OLID	CBASU
Glove	Formspring	OLID
W2V	Formspring	CBASU
Bert	CBASU	Formspring
ST	CBASU	Formspring
Bert	CBASU	OLID
ST	CBASU	OLID
Bert	OLID	Formspring
Bert	OLID	CBASU
Bert	Formspring	OLID
Bert	Formspring	CBASU

Table C.1: Cases Included for Comparison Between No Sampling and Normal Oversampling

Appendix D

List of Cases Included in ADASYN Oversampling and maGAN oversampling

Oversampling Method	**Embedding**	**Source**	**Target**
ADASYN40P	Glove	CBASU	OLID
ADASYN60P	Glove	CBASU	OLID
ADASYN40P	Glove	CBASU	Formspring
ADASYN60P	Glove	CBASU	Formspring
ADASYN10P	Glove	Formspring	OLID
ADASYN20P	Glove	Formspring	OLID
ADASYN30P	Glove	Formspring	OLID
ADASYN40P	Glove	Formspring	OLID
ADASYN50P	Glove	Formspring	OLID
maGAN40P	Glove	CBASU	OLID
maGAN60P	Glove	CBASU	OLID
maGAN80P	Glove	CBASU	OLID
maGAN40P	Glove	CBASU	Formspring
maGAN60P	Glove	CBASU	Formspring
maGAN80P	Glove	CBASU	Formspring

Table D.1: Cases using maGAN and ADASYN Oversampling Methods

Appendix E

Normality Test Results

Data Group	**Test Statistics**	**p Value**
All Data for Graph Construction Method 1, F1 Micro	0.9125	0.04013
All Data for Graph Construction Method 1, F1 Macro	0.9090	0.03366
All Data for Graph Construction Method 2, F1 Micro	0.9577	0.3942
All Data for Graph Construction Method 2, F1 Macro	0.9361	0.1337
SVM Poly Results, F1 Micro	0.7362	3.158×10^{-5}
SVM Poly Results, F1 Macro	0.7962	2.557×10^{-4}
SVM Rbf Results, F1 Micro	0.7432	1.394×10^{-5}
SVM Rbf Results, F1 Macro	0.8031	1.052×10^{-4}
Random Forest Results, F1 Micro	0.7432	3.972×10^{-5}
Random Forest Results, F1 Macro	0.8031	3.313×10^{-4}
ADDA Results, F1 Micro	0.5195	8.767×10^{-8}
ADDA Results, F1 Macro	0.5183	8.535×10^{-8}
GraphSAGE1 Results, F1 Micro	0.7834	1.599×10^{-4}
GraphSAGE1 Results, F1 Macro	0.9240	0.07189
GraphSAGE2 Results, F1 Micro	0.7663	8.706×10^{-5}
GraphSAGE2 Results, F1 Macro	0.7937	2.336×10^{-4}
FastGCN1 Results, F1 Micro	0.8033	3.345×10^{-4}
FastGCN1 Results, F1 Macro	0.9831	0.9457
FastGCN1d Results, F1 Micro	0.7792	1.372×10^{-4}
FastGCN1d Results, F1 Macro	0.6754	4.845×10^{-6}
FastGCN2 Results, F1 Micro	0.7834	1.599×10^{-4}
FastGCN2 Results, F1 Macro	0.9240	0.07189
FastGCN2d Results, F1 Micro	0.7663	8.706×10^{-5}
FastGCN2d Results, F1 Macro	0.7937	2.336×10^{-4}
Glove Results for Word Embedding Method Comparison, F1 Micro	0.8040	1.707×10^{-7}
Glove Results for Word Embedding Method Comparison, F1 Macro	0.9087	2.785×10^{-3}
W2V Results for Word Embedding Method Comparison, F1 Micro	0.6819	4.046×10^{-10}
W2V Results for Word Embedding Method Comparison, F1 Micro	0.8643	8.315×10^{-6}

ST Results for Word Embedding Method Comparison, F1 Micro	0.8049	1.792×10^{-7}
ST Results for Word Embedding Method Comparison, F1 Macro	0.7603	1.576×10^{-8}
Bert Results for Word Embedding Method Comparison, F1 Micro	0.7857	6.053×10^{-8}
Bert Results for Word Embedding Method Comparison, F1 Macro	0.8213	4.791×10^{-7}
Non-Oversampling Results for Oversampling Comparison, F1 Micro	0.7494	3.144×10^{-15}
Non-Oversampling Results for Oversampling Comparison, F1 Macro	0.8235	1.274×10^{-12}
1st Order Normal Oversampling Results for Oversampling Comparison, F1 Micro	0.7591	6.388×10^{-15}
1st Order Normal Oversampling Results for Oversampling Comparison, F1 Macro	0.8275	1.858×10^{-12}
2nd Order Normal Oversampling Results for Oversampling Comparison, F1 Micro	0.7374	1.348×10^{-15}
2nd Order Normal Oversampling Results for Oversampling Comparison, F1 Macro	0.8345	3.614×10^{-12}
3rd Order Normal Oversampling Results for Oversampling Comparison, F1 Micro	0.7807	3.411×10^{-13}
3rd Order Normal Oversampling Results for Oversampling Comparison, F1 Macro	0.8791	2.633×10^{-9}
ADASYN Oversampling Results for Oversampling Comparison, F1 Micro	0.7373	2.168×10^{-11}
ADASYN Oversampling Results for Oversampling Comparison, F1 Macro	0.8188	3.892×10^{-8}
maGAN Oversampling Results for Oversampling Comparison, F1 Micro	0.8135	1.313×10^{-5}
maGAN Oversampling Results for Oversampling Comparison, F1 Macro	0.8617	1.762×10^{-3}

Table E.1: Normality Test For All Data Groups

Appendix F

All Experiment Results

Embedding	Source-Target	Oversampling	Classifier	F1 Micro	F1 Macro
Glove	C-O	None	GraphSAGE1	0.64285	0.43563
			GraphSAGE2	0.61525	0.44859
			FastGCN1	0.66349	0.30919
			FastGCN1d	0.66712	0.40103
			FastGCN2	0.61910	0.28118
			FastGCN2d	0.62046	0.44709
			SVM,poly	0.70992	0.59047
			SVM,rbf	0.70924	0.56015
			Random Forest	0.69298	0.50643
			ADDA	0.6673	0.4002
Glove	C-F	None	GraphSAGE1	0.93345	0.48514
			GraphSAGE2	0.83052	0.46759
			FastGCN1	0.93912	0.45302
			FastGCN1d	0.93637	0.48603
			FastGCN2	0.92463	0.28118
			FastGCN2d	0.93062	0.48429
			SVM,poly	0.93243	0.66809
			SVM,rbf	0.93983	0.69473
			Random Forest	0.94266	0.63302
			ADDA	0.9391	0.4843
Glove	F-C	None	GraphSAGE1	0.81131	0.46825
			GraphSAGE2	0.78819	0.44607
			FastGCN1	0.80721	0.42236
			FastGCN1d	0.93684	0.48617
			FastGCN2	0.80746	0.33583
			FastGCN2d	0.80761	0.44678
			SVM,poly	0.81581	0.52589
			SVM,rbf	0.80760	0.44678

			Random Forest	0.80745	0.52518
			ADDA	0.8076	0.4468
Glove	F-O	None	GraphSAGE1	0.66500	0.40728
			GraphSAGE2	0.64912	0.44595
			FastGCN1	0.66765	0.46556
			FastGCN1d	0.66735	0.40090
			FastGCN2	0.61910	0.28118
			FastGCN2d	0.62046	0.44709
			SVM,poly	0.68315	0.46327
			SVM,rbf	0.66764	0.40122
			Random Forest	0.67241	0.45328
			ADDA	0.6673	0.4002
Glove	O-F	None	GraphSAGE1	0.66270	0.45224
			FastGCN1	0.12159	0.24464
			FastGCN1d	0.33777	0.29420
			SVM,poly	0.94109	0.60304
			SVM,rbf	0.93699	0.65718
			Random Forest	0.93754	0.63308
			ADDA	0.9391	0.4843
Glove	O-C	None	GraphSAGE1	0.56963	0.50324
			FastGCN1	0.40645	0.54676
			FastGCN1d	0.56943	0.46237
			SVM,poly	0.81341	0.64126
			SVM,rbf	0.80645	0.68000
			Random Forest	0.82111	0.67089
			ADDA	0.7951	0.5349
Glove	C-O	maGAN40P	GraphSAGE1	0.49433	0.48915
			GraphSAGE2	0.63014	0.46331
			FastGCN1	0.64126	0.28723
			FastGCN1d	0.59271	0.49993
			FastGCN2	0.65616	0.28923
			FastGCN2d	0.63612	0.46759
			SVM,poly	0.66477	0.58765
			SVM,rbf	0.70908	0.56190
			Random Forest	0.69328	0.50409
			ADDA	0.6305	0.4064
Glove	C-O	maGAN60P	GraphSAGE1	0.50197	0.48717
			GraphSAGE2	0.62326	0.46627
			FastGCN1	0.60216	0.67114
			FastGCN1d	0.57864	0.50664

			FastGCN2	0.62719	0.57799
			FastGCN2d	0.61283	0.47706
			SVM,poly	0.66787	0.58957
			SVM,rbf	0.70908	0.56213
			Random Forest	0.69419	0.50615
			ADDA	0.6248	0.4148
Glove	C-O	maGAN80P	GraphSAGE1	0.56292	0.50173
			GraphSAGE2	0.41016	0.40437
			FastGCN1	0.56549	0.56237
			FastGCN1d	0.58379	0.50131
			FastGCN2	0.57615	0.40115
			FastGCN2d	0.60345	0.47915
			SVM,poly	0.67460	0.59587
			SVM,rbf	0.71037	0.56685
			Random Forest	0.69517	0.50840
			ADDA	0.6430	0.4199
Glove	C-O	ADASYN40P	GraphSAGE1	0.54900	0.50780
			GraphSAGE2	0.57985	0.49344
			FastGCN1	0.64171	0.29022
			FastGCN1d	0.62931	0.47263
			FastGCN2	0.65207	0.54674
			FastGCN2d	0.61230	0.47332
			SVM,poly	0.71067	0.63169
			SVM,rbf	0.70924	0.56015
			Random Forest	0.70198	0.56638
			ADDA	0.6673	0.4002
Glove	C-O	ADASYN60P	GraphSAGE1	0.56299	0.50286
			GraphSAGE2	0.59929	0.48653
			FastGCN1	0.60390	0.29776
			FastGCN1d	0.60451	0.48695
			FastGCN2	0.63052	0.49989
			FastGCN2d	0.62008	0.46995
			SVM,poly	0.69993	0.64369
			SVM,rbf	0.71067	0.65521
			Random Forest	0.70477	0.60841
			ADDA	0.6673	0.4002
Glove	C-O	Normal40P	GraphSAGE1	0.59037	0.49869
			GraphSAGE2	0.64655	0.43042
			FastGCN1	0.63256	0.29680
			FastGCN1d	0.61222	0.48431
			FastGCN2	0.64292	0.64292
			FastGCN2d	0.62576	0.46788
			SVM,poly	0.71347	0.62349
			SVM,rbf	0.71347	0.62349
			Random Forest	0.71347	0.62349
			ADDA	0.6673	0.4002

Glove	C-O	Normal60P	GraphSAGE1	0.56518	0.50743
			GraphSAGE2	0.60851	0.48863
			FastGCN1	0.61199	0.27705
			FastGCN1d	0.61766	0.48154
			FastGCN2	0.61956	0.56648
			FastGCN2d	0.63286	0.45939
			SVM,poly	0.69222	0.64681
			SVM,rbf	0.71324	0.66864
			Random Forest	0.70062	0.56631
			ADDA	0.6552	0.4025
Glove	C-F	ADASYN40P	GraphSAGE1	0.86195	0.49378
			GraphSAGE2	0.92062	0.49376
			FastGCN1	0.83990	0.38040
			FastGCN1d	0.90770	0.52352
			FastGCN2	0.90873	0.50664
			FastGCN2d	0.92487	0.49074
			SVM,poly	0.86423	0.62656
			SVM,rbf	0.87124	0.64424
			Random Forest	0.91644	0.64593
			ADDA	0.9391	0.4843
Glove	C-F	ADASYN60P	GraphSAGE1	0.80753	0.50578
			GraphSAGE2	0.91928	0.49408
			FastGCN1	0.80832	0.31217
			FastGCN1d	0.81911	0.51204
			FastGCN2	0.90038	0.57260
			FastGCN2d	0.91794	0.48801
			SVM,poly	0.69222	0.64681
			SVM,rbf	0.71324	0.66864
			Random Forest	0.70062	0.56631
			ADDA	0.9391	0.4844
Glove	C-F	Normal40P	GraphSAGE1	0.59037	0.49869
			GraphSAGE2	0.64655	0.43042
			FastGCN1	0.63256	0.29680
			FastGCN1d	0.61222	0.48431
			FastGCN2	0.64292	0.64292
			FastGCN2d	0.62576	0.46788
			SVM,poly	0.71347	0.62349
			SVM,rbf	0.71347	0.62349
			Random Forest	0.71347	0.62349
			ADDA	0.9391	0.4843
Glove	C-F	Normal60P	GraphSAGE1	0.82714	0.51393
			GraphSAGE2	0.93401	0.48649
			FastGCN1	0.80903	0.28893
			FastGCN1d	0.81556	0.50925
			FastGCN2	0.88967	0.29680
			FastGCN2d	0.91085	0.50572
			SVM,poly	0.80059	0.58346
			SVM,rbf	0.78232	0.58240

			Random Forest	0.92250	0.65613
			ADDA	0.9391	0.4843
Glove	C-F	maGAN40P	GraphSAGE1	0.60702	0.41255
			GraphSAGE2	0.92487	0.49961
			FastGCN1	0.65616	0.28118
			FastGCN1d	0.63612	0.46759
			FastGCN2	0.89061	0.54675
			FastGCN2d	0.92314	0.49297
			SVM,poly	0.81296	0.59114
			SVM,rbf	0.93975	0.69368
			Random Forest	0.94306	0.63453
			ADDA	0.9391	0.4843
Glove	C-F	maGAN60P	GraphSAGE1	0.89872	0.57309
			GraphSAGE2	0.92629	0.49537
			FastGCN1	0.79201	0.49988
			FastGCN1d	0.81556	0.51784
			FastGCN2	0.85580	0.30974
			FastGCN2d	0.90006	0.50740
			SVM,poly	0.77925	0.57083
			SVM,rbf	0.93880	0.69457
			Random Forest	0.94329	0.63159
			ADDA	0.9391	0.4843
Glove	C-F	maGAN80P	GraphSAGE1	0.29532	0.26468
			GraphSAGE2	0.37171	0.30752
			FastGCN1	0.54631	0.34818
			FastGCN1d	0.71208	0.48324
			FastGCN2	0.54631	0.34818
			FastGCN2d	0.75516	0.49992
			SVM,poly	0.75744	0.55983
			SVM,rbf	0.93747	0.69114
			Random Forest	0.94251	0.62718
			ADDA	0.0609	0.0574
Glove	F-O	ADASYN10P	GraphSAGE1	0.66735	0.40024
			GraphSAGE2	0.63370	0.46041
			FastGCN1	0.66720	0.40019
			FastGCN1d	0.66720	0.40019
			FastGCN2	0.66674	0.28711
			FastGCN2d	0.66697	0.40271
			SVM,poly	0.69404	0.51262
			SVM,rbf	0.67445	0.42328
			Random Forest	0.67052	0.41632
			ADDA	0.6673	0.4002
Glove	F-O	ADASYN20P	GraphSAGE1	0.60027	0.48262
			GraphSAGE2	0.60428	0.48795
			FastGCN1	0.66485	0.26494
			FastGCN1d	0.66546	0.40619
			FastGCN2	0.66682	0.26647
			FastGCN2d	0.65449	0.42585

			SVM,poly	0.70757	0.57905
			SVM,rbf	0.69517	0.49426
			Random Forest	0.66893	0.40559
			ADDA	0.6673	0.4002
Glove	F-O	ADASYN30P	GraphSAGE1	0.57033	0.49601
			GraphSAGE2	0.65563	0.43775
			FastGCN1	0.64806	0.30817
			FastGCN1d	0.62371	0.47010
			FastGCN2	0.66553	0.26556
			FastGCN2d	0.64844	0.42891
			SVM,poly	0.71355	0.60353
			SVM,rbf	0.69820	0.50718
			Random Forest	0.66855	0.40415
			ADDA	0.6670	0.4010
Glove	F-O	ADASYN40P	GraphSAGE1	0.54462	0.49702
			GraphSAGE2	0.54688	0.54688
			FastGCN1	0.62954	0.47382
			FastGCN1d	0.60345	0.47609
			FastGCN2	0.54688	0.51581
			FastGCN2d	0.64209	0.64209
			SVM,poly	0.71082	0.60934
			SVM,rbf	0.69820	0.50943
			Random Forest	0.66870	0.40464
			ADDA	0.6671	0.4004
Glove	F-O	ADASYN50P	GraphSAGE1	0.66470	0.40443
			GraphSAGE2	0.51550	0.49288
			FastGCN1	0.59770	0.32714
			FastGCN1d	0.58507	0.48312
			FastGCN2	0.65434	0.30057
			FastGCN2d	0.64201	0.44845
			SVM,poly	0.71000	0.61591
			SVM,rbf	0.69850	0.51450
			Random Forest	0.66863	0.40440
			ADDA	0.6673	0.4002
Glove	F-O	Normal10P	GraphSAGE1	0.63506	0.45563
			GraphSAGE2	0.63649	0.45799
			FastGCN1	0.66697	0.23432
			FastGCN1d	0.66697	0.40314
			FastGCN2	0.66720	0.29421
			FastGCN2d	0.66289	0.40853
			SVM,poly	0.69207	0.52662
			SVM,rbf	0.68315	0.45409
			Random Forest	0.67196	0.42283
			ADDA	0.6673	0.4002
			GraphSAGE1	0.62349	0.46690
			GraphSAGE2	0.60670	0.48645
			FastGCN1	0.66485	0.29398
			FastGCN1d	0.66583	0.40379
Glove	F-O	Normal20P			

			FastGCN2	0.66727	0.67935
			FastGCN2d	0.66077	0.41710
			SVM,poly	0.69630	0.58004
			SVM,rbf	0.71052	0.56731
			Random Forest	0.66764	0.40122
			ADDA	0.6673	0.4002
Glove	F-O	Normal30P	GraphSAGE1	0.57328	0.49498
			GraphSAGE2	0.66735	0.40024
			FastGCN1	0.64080	0.28095
			FastGCN1d	0.62613	0.45488
			FastGCN2	0.66485	0.25666
			FastGCN2d	0.65162	0.43076
			SVM,poly	0.69895	0.59742
			SVM,rbf	0.70992	0.58798
			Random Forest	0.66757	0.40098
			ADDA	0.6673	0.4002
Glove	F-O	Normal40P	GraphSAGE1	0.54658	0.49794
			GraphSAGE2	0.54454	0.49839
			FastGCN1	0.63596	0.51861
			FastGCN1d	0.61124	0.47442
			FastGCN2	0.66047	0.30419
			FastGCN2d	0.64224	0.44888
			SVM,poly	0.70183	0.60901
			SVM,rbf	0.71166	0.60374
			Random Forest	0.66757	0.40098
			ADDA	0.6670	0.4005
Glove	F-O	Normal50P	GraphSAGE1	0.65048	0.42706
			GraphSAGE2	0.51626	0.49326
			FastGCN1	0.60859	0.28756
			FastGCN1d	0.58379	0.48017
			FastGCN2	0.65850	0.26556
			FastGCN2d	0.63203	0.46716
			SVM,poly	0.69850	0.61560
			SVM,rbf	0.71355	0.61374
			Random Forest	0.66749	0.40073
			ADDA	0.4756	0.4565
W2V	C-O	None	GraphSAGE1	0.59304	0.49151
			GraphSAGE2	0.62429	0.47062
			FastGCN1	0.66258	0.31243
			FastGCN1d	0.66583	0.40229
			FastGCN2	0.66356	0.35929
			FastGCN2d	0.66553	0.40154
			SVM,poly	0.71517	0.58032
			SVM,rbf	0.71714	0.58395
			Random Forest	0.68543	0.46148
			ADDA	0.6673	0.4002

W2V	C-F	None	GraphSAGE1	0.86141	0.51233
			GraphSAGE2	0.84478	0.47409
			FastGCN1	0.93779	0.76532
			FastGCN1d	0.93430	0.49362
			FastGCN2	0.92417	0.32313
			FastGCN2d	0.93058	0.48765
			SVM,poly	0.93549	0.67909
			SVM,rbf	0.93525	0.67768
			Random Forest	0.94024	0.64821
			ADDA	0.7253	0.4416
W2V	F-C	None	GraphSAGE1	0.82141	0.54956
			GraphSAGE2	0.79508	0.44728
			FastGCN1	0.80786	0.45478
			FastGCN1d	0.80573	0.47161
			FastGCN2	0.80512	0.64048
			FastGCN2d	0.80502	0.44599
			SVM,poly	0.83130	0.59382
			SVM,rbf	0.81131	0.48297
			Random Forest	0.80649	0.45509
			ADDA	0.8051	0.4460
W2V	F-O	None	GraphSAGE1	0.64646	0.44284
			GraphSAGE2	0.65123	0.44267
			FastGCN1	0.66727	0.28118
			FastGCN1d	0.66727	0.28118
			FastGCN2	0.66742	0.28118
			FastGCN2d	0.66682	0.40071
			SVM,poly	0.67370	0.42134
			SVM,rbf	0.66863	0.40462
			Random Forest	0.66742	0.40114
			ADDA	0.6673	0.4002
W2V	O-F	None	GraphSAGE1	0.86964	0.49839
			GraphSAGE2	0.79658	0.47398
			FastGCN1	0.77917	0.37491
			FastGCN1d	0.79460	0.48353
			FastGCN2	0.77940	0.27275
			FastGCN2d	0.81756	0.49217
			SVM,poly	0.86378	0.62454
			SVM,rbf	0.86979	0.62945
			Random Forest	0.89187	0.63741
			ADDA	0.8606	0.4889
W2V	O-C	None	GraphSAGE1	0.79817	0.48625
			GraphSAGE2	0.60071	0.44333
			FastGCN1	0.79655	0.51462
			FastGCN1d	0.78234	0.48355
			FastGCN2	0.75018	0.34526
			FastGCN2d	0.78503	0.46472
			SVM,poly	0.77625	0.71034
			SVM,rbf	0.76763	0.70235

			Random Forest	0.82227	0.71197
			ADDA	0.8055	0.4461
W2V	C-F	Normal40P	GraphSAGE1	0.75336	0.48939
			GraphSAGE2	0.83196	0.48683
			FastGCN1	0.86932	0.98735
			FastGCN1d	0.84494	0.51530
			FastGCN2	0.85088	0.42236
			FastGCN2d	0.82911	0.49908
			SVM,poly	0.90984	0.66159
			SVM,rbf	0.90216	0.66008
			Random Forest	0.92021	0.65317
			ADDA	0.7416	0.4907
W2V	C-F	Normal60P	GraphSAGE1	0.72406	0.49974
			GraphSAGE2	0.79642	0.48859
			FastGCN1	0.80505	0.72406
			FastGCN1d	0.81407	0.50927
			FastGCN2	0.78265	0.58904
			FastGCN2d	0.78091	0.48941
			SVM,poly	0.87517	0.63493
			SVM,rbf	0.83955	0.61296
			Random Forest	0.88618	0.62915
			ADDA	0.6992	0.4648
W2V	C-F	Normal80P	GraphSAGE1	0.91760	0.50139
			GraphSAGE2	0.52382	0.38658
			FastGCN1	0.74149	0.68719
			FastGCN1d	0.76904	0.49664
			FastGCN2	0.76405	0.89823
			FastGCN2d	0.76120	0.47272
			SVM,poly	0.84779	0.61441
			SVM,rbf	0.79040	0.57983
			Random Forest	0.84304	0.60163
			ADDA	0.7367	0.4783
W2V	C-O	Normal40P	GraphSAGE1	0.61937	0.47086
			GraphSAGE2	0.52123	0.49357
			FastGCN1	0.63557	0.87865
			FastGCN1d	0.62777	0.46105
			FastGCN2	0.64253	0.46864
			FastGCN2d	0.63859	0.45977
			SVM,poly	0.72281	0.61925
			SVM,rbf	0.73060	0.64453
			Random Forest	0.69572	0.50354
			ADDA	0.6349	0.4634
W2V	C-O	Normal60P	GraphSAGE1	0.57510	0.50095
			GraphSAGE2	0.57374	0.49255
			FastGCN1	0.58017	0.85917
			FastGCN1d	0.56406	0.49716
			FastGCN2	0.58456	0.37862
			FastGCN2d	0.60189	0.49128

			SVM,poly	0.72463	0.64128
			SVM,rbf	0.73174	0.67647
			Random Forest	0.70715	0.54990
			ADDA	0.5658	0.4615
W2V	C-O	Normal80P	GraphSAGE1	0.55551	0.49664
			GraphSAGE2	0.54037	0.49332
			FastGCN1	0.56542	0.37701
			FastGCN1d	0.55785	0.49482
			FastGCN2	0.58903	0.70569
			FastGCN2d	0.54612	0.49406
			SVM,poly	0.72364	0.64941
			SVM,rbf	0.72811	0.68554
			Random Forest	0.71161	0.57977
			ADDA	0.5915	0.5107
W2V	O-C	Normal60P	GraphSAGE1	0.79706	0.48299
			GraphSAGE2	0.47128	0.38725
			FastGCN1	0.69401	0.68787
			FastGCN1d	0.93383	0.49112
			FastGCN2	0.79848	0.55963
			FastGCN2d	0.57494	0.43329
			SVM,poly	0.76463	0.70230
			SVM,rbf	0.75124	0.69157
			Random Forest	0.81674	0.71485
			ADDA	0.6229	0.5095
W2V	O-C	Normal80P	GraphSAGE1	0.78432	0.47993
			GraphSAGE2	0.52400	0.42091
			FastGCN1	0.74871	0.52083
			FastGCN1d	0.59929	0.49567
			FastGCN2	0.76535	0.39585
			FastGCN2d	0.64911	0.43497
			SVM,poly	0.74997	0.69288
			SVM,rbf	0.72739	0.67586
			Random Forest	0.80837	0.71999
			ADDA	0.7083	0.4730
W2V	O-C	Normal100P	GraphSAGE1	0.73186	0.52501
			GraphSAGE2	0.34044	0.30846
			FastGCN1	0.69853	0.35929
			FastGCN1d	0.70198	0.58383
			FastGCN2	0.47555	0.80795
			FastGCN2d	0.55687	0.42056
			SVM,poly	0.73566	0.68182
			SVM,rbf	0.70669	0.66099
			Random Forest	0.79218	0.71319
			ADDA	0.7769	0.5048
			GraphSAGE1	0.83164	0.48423
			GraphSAGE2	0.87051	0.49458
			FastGCN1	0.93913	0.68622
			FastGCN1d	0.93383	0.49112
W2V	O-F	Normal60P			

			FastGCN2	0.78186	0.51655
			FastGCN2d	0.82041	0.48515
			SVM,poly	0.84755	0.61345
			SVM,rbf	0.84185	0.60921
			Random Forest	0.87304	0.62430
			ADDA	0.8885	0.5053
W2V	O-F	Normal80P	GraphSAGE1	0.68569	0.46513
			GraphSAGE2	0.85753	0.49703
			FastGCN1	0.68054	0.35963
			FastGCN1d	0.69345	0.46852
			FastGCN2	0.73516	0.32176
			FastGCN2d	0.72519	0.47163
			SVM,poly	0.81890	0.59326
			SVM,rbf	0.79768	0.58036
			Random Forest	0.82562	0.58985
			ADDA	0.8641	0.5053
W2V	O-F	Normal100P	GraphSAGE1	0.69353	0.46761
			GraphSAGE2	0.74331	0.48345
			FastGCN1	0.70120	0.38062
			FastGCN1d	0.69930	0.46992
			FastGCN2	0.71529	0.30959
			FastGCN2d	0.70057	0.46705
			SVM,poly	0.79515	0.57730
			SVM,rbf	0.75945	0.55541
			Random Forest	0.78731	0.56307
			ADDA	0.3585	0.3091
W2V	F-C	Normal10P	GraphSAGE1	0.79548	0.52705
			GraphSAGE2	0.79234	0.45030
			FastGCN1	0.80700	0.37491
			FastGCN1d	0.80736	0.46111
			FastGCN2	0.80472	0.67313
			FastGCN2d	0.80518	0.44987
			SVM,poly	0.84312	0.65906
			SVM,rbf	0.84023	0.65000
			Random Forest	0.80750	0.46116
			ADDA	0.8051	0.4460

W2V	F-C	Normal20P	GraphSAGE1	0.81380	0.53000
			GraphSAGE2	0.79914	0.45799
			FastGCN1	0.81644	0.62484
			FastGCN1d	0.81639	0.52328
			FastGCN2	0.80304	0.75684
			FastGCN2d	0.80497	0.47838
			SVM,poly	0.84956	0.69844
			SVM,rbf	0.84277	0.84277
			Random Forest	0.80969	0.47338
			ADDA	0.6144	0.4390
W2V	F-C	Normal30P	GraphSAGE1	0.77763	0.50669
			GraphSAGE2	0.77123	0.47651
			FastGCN1	0.80411	0.58593
			FastGCN1d	0.78732	0.49105
			FastGCN2	0.80639	0.50927
			FastGCN2d	0.80644	0.45558
			SVM,poly	0.84936	0.70804
			SVM,rbf	0.84474	0.73803
			Random Forest	0.81146	0.81146
			ADDA	0.7202	0.4780
W2V	F-C	Normal40P	GraphSAGE1	0.79691	0.49878
			GraphSAGE2	0.73009	0.45381
			FastGCN1	0.79827	0.45711
			FastGCN1d	0.74937	0.45530
			FastGCN2	0.80888	0.56532
			FastGCN2d	0.80472	0.50140
			SVM,poly	0.84977	0.71228
			SVM,rbf	0.84165	0.74025
			Random Forest	0.81319	0.49404
			ADDA	0.7825	0.4521
ST	C-O	None	GraphSAGE1	0.66503	0.40199
			GraphSAGE2	0.57160	0.50127
			FastGCN1	0.65566	0.58859
			FastGCN1d	0.66405	0.41468
			FastGCN2	0.66767	0.71886
			FastGCN2d	0.66767	0.40036
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66895	0.41505
			ADDA	0.6677	0.4004
ST	C-F	None	GraphSAGE1	0.90652	0.49872
			GraphSAGE2	0.66640	0.42067
			FastGCN1	0.93912	0.56967
			FastGCN1d	0.93912	0.48430
			FastGCN2	0.93590	0.60967
			FastGCN2d	0.93763	0.48391
			SVM,poly	0.93912	0.48430

			SVM,rbf	0.93912	0.48430
			Random Forest	0.93203	0.52477
			ADDA	0.9391	0.4843
ST	F-C	None	GraphSAGE1	0.79773 0.45773	
			GraphSAGE2	0.80618	0.44635
			FastGCN1	0.80768 0.64047	
			FastGCN1d	0.80768	0.44681
			FastGCN2	0.80768 0.78141	
			FastGCN2d	0.80768	0.44681
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.80703 0.45142	
			ADDA	0.8077	0.4468
ST	F-O	None	GraphSAGE1	0.66767	0.40036
			GraphSAGE2	0.64977	0.44381
			FastGCN1	0.66767	0.53366
			FastGCN1d	0.66767	0.40036
			FastGCN2	0.66767	0.46901
			FastGCN2d	0.66767	0.40036
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66971	0.40844
			ADDA	0.6677	0.4004
ST	O-C	None	GraphSAGE1	0.82674	0.64045
			GraphSAGE2	0.66718	0.46146
			FastGCN1	0.80768	0.67686
			FastGCN1d	0.80768	0.44681
			FastGCN2	0.39369	0.70238
			FastGCN2d	0.80768	0.44681
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.77537	0.52927
			ADDA	0.8077	0.4468
ST	O-F	None	GraphSAGE1	0.92014	0.49815
			GraphSAGE2	0.93408	0.49233
			FastGCN1	0.93912	0.61215
			FastGCN1d	0.74807	0.49724
			FastGCN2	0.93133	0.55078
			FastGCN2d	0.93912	0.48430
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.93912	0.48430
			Random Forest	0.91093	0.52740
			ADDA	0.9391	0.4843

ST	C-O	Normal40P	GraphSAGE1	0.50642	0.48723
			GraphSAGE2	0.50347	0.48761
			FastGCN1	0.56427	0.92171
			FastGCN1d	0.53618	0.53618
			FastGCN2	0.54766	0.86177
			FastGCN2d	0.60921	0.48093
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66925	0.44403
			ADDA	0.6677	0.4004
ST	C-O	Normal60P	GraphSAGE1	0.55952	0.49375
			GraphSAGE2	0.61548	0.47810
			FastGCN1	0.54260	0.89984
			FastGCN1d	0.56964	0.49865
			FastGCN2	0.61073	0.92166
			FastGCN2d	0.62402	0.46527
			SVM,poly	0.61087	0.52965
			SVM,rbf	0.65929	0.50420
			Random Forest	0.66578	0.49490
			ADDA	0.6677	0.4004
ST	C-O	Normal80P	GraphSAGE1	0.62266	0.47845
			GraphSAGE2	0.59569	0.49064
			FastGCN1	0.60823	0.91538
			FastGCN1d	0.62024	0.47876
			FastGCN2	0.59517	0.93756
			FastGCN2d	0.60695	0.47751
			SVM,poly	0.51676	0.51314
			SVM,rbf	0.54154	0.53148
			Random Forest	0.65407	0.53433
			ADDA	0.6677	0.4004
ST	C-F	Normal40P	GraphSAGE1	0.72444	0.47648
			GraphSAGE2	0.91928	0.49408
			FastGCN1	0.75358	0.45883
			FastGCN1d	0.83045	0.50958
			FastGCN2	0.71279	0.76089
			FastGCN2d	0.72783	0.47455
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.93912	0.48430
			Random Forest	0.91093	0.56338
			ADDA	0.9391	0.4843
ST	C-F	Normal60P	GraphSAGE1	0.86620	0.50547
			GraphSAGE2	0.92172	0.49801
			FastGCN1	0.76398	0.65227
			FastGCN1d	0.77894	0.50387
			FastGCN2	0.84242	0.64629
			FastGCN2d	0.85848	0.50486
			SVM,poly	0.69294	0.48941
			SVM,rbf	0.75909	0.51449

			Random Forest	0.85359 0.54979	
			ADDA	0.9391	0.4843
ST	C-F	Normal80P	GraphSAGE1	0.77595	0.50315
			GraphSAGE2	0.90093	0.50022
			FastGCN1	0.70452	0.72805
			FastGCN1d	0.70153	0.47718
			FastGCN2	0.83194	0.63609
			FastGCN2d	0.82052	0.48692
			SVM,poly	0.43684	0.36058
			SVM,rbf	0.39376	0.33399
			Random Forest	0.76437	0.51715
			ADDA	0.0625	0.0593
Bert	C-O	None	GraphSAGE1	0.66692	0.40355
			GraphSAGE2	0.66571	0.40310
			FastGCN1	0.66631	0.74394
			FastGCN1d	0.66767	0.40145
			FastGCN2	0.66767	0.63297
			FastGCN2d	0.66344	0.41119
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6677	0.4004
Bert	C-F	None	GraphSAGE1	0.87730	0.50346
			GraphSAGE2	0.85439	0.48085
			FastGCN1	0.92007	0.62485
			FastGCN1d	0.93519	0.48687
			FastGCN2	0.91085	0.70618
			FastGCN2d	0.93227	0.49047
			SVM,poly	0.93951	0.49708
			SVM,rbf	0.93936	0.49203
			Random Forest	0.93912	0.48430
			ADDA	0.9391	0.4843
Bert	F-C	None	GraphSAGE1	0.79878	0.51186
			GraphSAGE2	0.92838	0.60735
			FastGCN1	0.80763	0.64351
			FastGCN1d	0.80768	0.44681
			FastGCN2	0.77457	0.64926
			FastGCN2d	0.80723	0.44744
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.80768	0.44680
			ADDA	0.8077	0.4468
Bert	F-C	None	GraphSAGE1	0.79878	0.51186
			GraphSAGE2	0.92838	0.60735
			FastGCN1	0.80763	0.64351
			FastGCN1d	0.80768	0.44681
			FastGCN2	0.77457	0.64926
			FastGCN2d	0.80723	0.44744

			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.80768	0.44680
			ADDA	0.8077	0.4468
Bert	F-O	None	GraphSAGE1	0.66495	0.40515
			GraphSAGE2	0.66707	0.40058
			FastGCN1	0.66767	0.54932
			FastGCN1d	0.66767	0.40037
			FastGCN2	0.66752	0.46864
			FastGCN2d	0.66692	0.40333
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6677	0.4004
Bert	O-C	None	GraphSAGE1	0.77117	0.60003
			GraphSAGE2	0.78092	0.49401
			FastGCN1	0.80776	0.91041
			FastGCN1d	0.77947	0.57407
			FastGCN2	0.59751	0.65619
			FastGCN2d	0.79698	0.44689
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.57580	0.49036
			ADDA	0.7453	0.4638
Bert	O-F	None	GraphSAGE1	0.93109	0.48443
			GraphSAGE2	0.92007	0.49536
			FastGCN1	0.85840	0.49988
			FastGCN1d	0.92156	0.49324
			FastGCN2	0.72350	0.57799
			FastGCN2d	0.84675	0.50264
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.93912	0.48430
			Random Forest	0.63443	0.43905
			ADDA	0.8256	0.4801
Bert	C-O	Normal40P	GraphSAGE1	0.63421	0.46104
			GraphSAGE2	0.66767	0.40477
			FastGCN1	0.62304	0.59401
			FastGCN1d	0.56352	0.49957
			FastGCN2	0.65816	0.60306
			FastGCN2d	0.59222	0.48492
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6313	0.4542
			GraphSAGE1	0.62545	0.47385
			GraphSAGE2	0.65801	0.42353
			FastGCN1	0.62953	0.65816
			FastGCN1d	0.63014	0.47004
Bert	C-O	Normal60P			

			FastGCN2	0.65974	0.72677
			FastGCN2d	0.57545	0.49678
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6210	0.4621
Bert	C-O	Normal80P	GraphSAGE1	0.66110	0.42416
			GraphSAGE2	0.64003	0.45821
			FastGCN1	0.62485	0.77408
			FastGCN1d	0.62500	0.47417
			FastGCN2	0.64071	0.88771
			FastGCN2d	0.64086	0.45753
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.5115	0.4735
Bert	C-O	Normal100P	GraphSAGE1	0.62530	0.47471
			GraphSAGE2	0.66095	0.41833
			FastGCN1	0.62326	0.70296
			FastGCN1d	0.62296	0.47656
			FastGCN2	0.63995	0.85554
			FastGCN2d	0.64018	0.45915
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.5330	0.4902
Bert	C-F	Normal40P	GraphSAGE1	0.75500	0.48217
			GraphSAGE2	0.84580	0.48644
			FastGCN1	0.67396	0.82914
			FastGCN1d	0.78123	0.49020
			FastGCN2	0.78776	0.47560
			FastGCN2d	0.82076	0.99905
			SVM,poly	0.93439	0.51450
			SVM,rbf	0.93439	0.51450
			Random Forest	0.93912	0.48430
			ADDA	0.7728	0.4476
Bert	C-F	Normal60P	GraphSAGE1	0.76177	0.47497
			GraphSAGE2	0.74398	0.48793
			FastGCN1	0.70145	0.67172
			FastGCN1d	0.73744	0.47144
			FastGCN2	0.66318	0.64592
			FastGCN2d	0.71397	0.47449
			SVM,poly	0.91612	0.53239
			SVM,rbf	0.87785	0.53606
			Random Forest	0.93912	0.48430
			ADDA	0.7808	0.4608

Bert	C-F	Normal80P	GraphSAGE1	0.65207	0.45056
			GraphSAGE2	0.74090	0.46157
			FastGCN1	0.55434	0.69507
			FastGCN1d	0.64357	0.44624
			FastGCN2	0.52874	0.90471
			FastGCN2d	0.52552	0.38659
			SVM,poly	0.90171	0.53453
			SVM,rbf	0.78122	0.50652
			Random Forest	0.93912	0.48559
			ADDA	0.7499	0.4509
Bert	C-F	Normal100P	GraphSAGE1	0.51788	0.38189
			GraphSAGE2	0.57702	0.40771
			FastGCN1	0.57166	0.87422
			FastGCN1d	0.57277	0.40949
			FastGCN2	0.57166	0.76000
			FastGCN2d	0.57277	0.40949
			SVM,poly	0.89525	0.53522
			SVM,rbf	0.70515	0.48051
			Random Forest	0.93920	0.48817
			ADDA	0.5665	0.4189
Bert	F-C	Normal10P	GraphSAGE1	0.79623	0.53454
			GraphSAGE2	0.73766	0.45826
			FastGCN1	0.80203	0.68555
			FastGCN1d	0.80733	0.46307
			FastGCN2	0.76217	0.62485
			FastGCN2d	0.80253	0.44969
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.80768	0.44680
			ADDA	0.8077	0.4468
Bert	F-C	Normal20P	GraphSAGE1	0.77802	0.53700
			GraphSAGE2	0.71065	0.45039
			FastGCN1	0.79083	0.71222
			FastGCN1d	0.79563	0.49411
			FastGCN2	0.74396	0.64047
			FastGCN2d	0.75011	0.47690
			SVM,poly	0.80778	0.44990
			SVM,rbf	0.80763	0.44883
			Random Forest	0.80768	0.44680
			ADDA	0.8077	0.4468
Bert	F-C	Normal30P	GraphSAGE1	0.68299	0.51784
			GraphSAGE2	0.71545	0.49592
			FastGCN1	0.72260	0.77186
			FastGCN1d	0.79853	0.46599
			FastGCN2	0.64633	0.65886
			FastGCN2d	0.79068	0.48407
			SVM,poly	0.80823	0.45358
			SVM,rbf	0.80773	0.46845

			Random Forest	0.80768	0.44680
			ADDA	0.7107	0.4499
Bert	F-C	Normal40P	GraphSAGE1	0.72730	0.44629
			GraphSAGE2	0.65608	0.47959
			FastGCN1	0.63907	0.65786
			FastGCN1d	0.79058	0.45725
			FastGCN2	0.60016	0.67026
			FastGCN2d	0.65843	0.47025
			SVM,poly	0.80833	0.45462
			SVM,rbf	0.79942	0.48510
			Random Forest	0.80768	0.44680
			ADDA	0.8050	0.4462
Bert	F-O	Normal10P	GraphSAGE1	0.66752	0.40031
			GraphSAGE2	0.66609	0.40303
			FastGCN1	0.66767	0.54446
			FastGCN1d	0.66767	0.40036
			FastGCN2	0.66752	0.53376
			FastGCN2d	0.66760	0.40034
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6520	0.4319
Bert	F-O	Normal20P	GraphSAGE1	0.66760	0.40034
			GraphSAGE2	0.66730	0.40088
			FastGCN1	0.66684	0.55755
			FastGCN1d	0.65514	0.42829
			FastGCN2	0.66737	0.52063
			FastGCN2d	0.66745	0.40115
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6400	0.4473
Bert	F-O	Normal30P	GraphSAGE1	0.66752	0.40096
			GraphSAGE2	0.66699	0.40250
			FastGCN1	0.65438	0.55023
			FastGCN1d	0.63724	0.45044
			FastGCN2	0.65937	0.54844
			FastGCN2d	0.64811	0.43788
			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6666	0.4006
Bert	F-O	Normal40P	GraphSAGE1	0.66669	0.40282
			GraphSAGE2	0.66548	0.40365
			FastGCN1	0.63927	0.51142
			FastGCN1d	0.58444	0.49624
			FastGCN2	0.66292	0.57158
			FastGCN2d	0.64101	0.44387

			SVM,poly	0.66767	0.40036
			SVM,rbf	0.66767	0.40036
			Random Forest	0.66767	0.40036
			ADDA	0.6594	0.4213
Bert	O-C	Normal60P	GraphSAGE1	0.79948	0.58457
			GraphSAGE2	0.79278	0.51947
			FastGCN1	0.81293	0.74652
			FastGCN1d	0.85943	0.49055
			FastGCN2	0.62509	0.71858
			FastGCN2d	0.73020	0.49171
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.80768	0.44680
			Random Forest	0.76821	0.48831
			ADDA	0.7667	0.4706
Bert	O-C	Normal80P	GraphSAGE1	0.87456	0.74290
			GraphSAGE2	0.78202	0.55842
			FastGCN1	0.79348	0.86317
			FastGCN1d	0.77432	0.54413
			FastGCN2	0.73856	0.68359
			FastGCN2d	0.81403	0.48198
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.46006	0.41885
			Random Forest	0.80503	0.45051
			ADDA	0.7361	0.4366
Bert	O-C	Normal100P	GraphSAGE1	0.87256	0.73305
			GraphSAGE2	0.83119	0.58819
			FastGCN1	0.75011	0.88755
			FastGCN1d	0.76992	0.50618
			FastGCN2	0.67384	0.97587
			FastGCN2d	0.73116	0.48019
			SVM,poly	0.80768	0.44680
			SVM,rbf	0.42354	0.39661
			Random Forest	0.80763	0.44679
			ADDA	0.6629	0.4784
Bert	O-F	Normal60P	GraphSAGE1	0.92416	0.50957
			GraphSAGE2	0.92763	0.50005
			FastGCN1	0.90267	0.70475
			FastGCN1d	0.91211	0.51501
			FastGCN2	0.75193	0.99053
			FastGCN2d	0.85746	0.49397
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.93912	0.48430
			Random Forest	0.82831	0.50288
			ADDA	0.9178	0.4987
			GraphSAGE1	0.84257	0.51474
			GraphSAGE2	0.91715	0.49755
			FastGCN1	0.74319	0.51093
			FastGCN1d	0.79642	0.51495
Bert	O-F	Normal80P			

			FastGCN2	0.73185	0.52659
			FastGCN2d	0.80918	0.50421
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.49692	0.38556
			Random Forest	0.93770	0.48392
			ADDA	0.9391	0.4843
Bert	O-F	Normal100P	GraphSAGE1	0.83438	0.52203
			GraphSAGE2	0.90148	0.51573
			FastGCN1	0.80832	0.52061
			FastGCN1d	0.81344	0.50895
			FastGCN2	0.77004	0.98350
			FastGCN2d	0.83399	0.51318
			SVM,poly	0.93912	0.48430
			SVM,rbf	0.44337	0.35700
			Random Forest	0.93888	0.48424
			ADDA	0.1833	0.1799

Table F.1: All Experiment Results