Aras Fırat Ünal
Batu Arda Düzgün
Burak Susam

SPAM E-MAIL CLASSIFICATION WITH GRAPH NEURAL NETWORKS

Abstract: *Email has become an essential cornerstone of everyone's life, and because of this spam mails have become a time waster and even a potential danger for every member of society. This problem has been combated with a variety of methods over the years and machine learning approaches have been the dominant method in recent years. Graph neural networks are a natural fit for this task as languages and sentences are inherently connected nodes that affect each other and GNN works well in dealing with complex structures and preserving global information. In this project, we go over the history of spam ham classification then use a simple yet elegant graph neural network design on a variety of data sets and show that graph neural networks are very efficient for spam ham classification. Additionally, the experiments show that our simpler algorithm performs better than a more complicated graph neural network and in time the pattern of spam mails change.*

1. Introduction

Due to the large amount of data going around on the internet, text classification is a popular problem for internet users. Especially for the management of the mailboxes, regulations of the social networks such as Facebook and Twitter, it has been quite important for both individual users and the communities. Conventional machine learning models have been widely used for classification purposes and they have been quite successful in terms of keeping communities regulated.

Because of the large amounts of data that is still enlarging day by day, the necessity of more efficient and more precise algorithms emerged. With the developing and expanding application area of graph neural networks (GNN), they engaged in natural language processing frameworks [1]. By representing the data as signals, the data can be encoded into a graph. Since understanding the relations between the parts in a text data is quite fundamental to classify it, graph neural networks are very suitable for this kind of task as they are utilized in capturing rich relational structures and preserving the global structural information of graphs [2], [3].

One of the sub-classes of text classification is spam-ham classification. Nowadays, the number of text and email messages that are being sent is increasing rapidly due to the increase in the usage of internet services by the banks to governments to social media platforms. It is important to classify these messages, in particular for this research emails, as spam or ham for both saving users time and protecting them from malicious phishing emails [4], [5], [6], [7]. Due to publicly available datasets such as Enron Spam dataset and datasets from Kaggle the methods regarding the text summarization can be tested with real emails hand labeled as ham and spam [8], [9], [10].

Graph neural networks are widely used in classification tasks. Especially, by using their ability to establish connections and find relations between different nodes, these networks are quite applicable to classification problems. In the light of this, GNN's are selected as the topic of this research project. For the graph structured data, the model can be scaled with the number of edges in the graph and hidden layers can be learned afterwards. [11] Convolving graph structured data is fairly straight-forward, but processing data in the way that it can be represented with graph structure is the key here. In NLP applications, choosing an extractive model to take the data as words or small word groups eases the process.

A short summary of the remainder of the paper would be: Related Works will go over the history of spam ham classification and previous works on GNN models will be given. Then, Methods will be the description of the model and datasets. In the Results section tables and figures for these models can be seen. In the Discussion and Conclusion part, the project is compared with other models and the results are interpreted. The code for the research project is added in the Appendix's Code Section.

2. Related Works

Because of the more sophisticated attack strategies that arise from time to time, automated spam email detection has become a difficult task. Several anti-spam systems have been presented throughout the years to stop unwanted email messages sent via the internet. Acts to control email communication [12], [13], email protocols [14], [15], refining email protocol control policies, address protection [16], list-based systems and keyword filtering [17], [18], [19], [20], challenge-response (CR) systems [21], collaborative spam filtering [22], and Honeypots [23] are all used in these systems. The self-learning system uses supervised and unsupervised machine learning algorithms to learn the behavior of spam and valid emails and automatically identify them without the need for human intervention [24], [25], [26], [27], [28]. Furthermore, machine learning-based systems are capable of fast adapting to the extremely dynamic character of spam emails [29]. We will be going over these different approaches in the following paragraphs.

As discussed in [12], [13] The US, Canada, Australia, and the European Union, including the United Kingdom, have all enacted legislation to prevent Unsolicited Commercial Communication (UCE) around 2005. UCE is commonly referred to as spam and UCE is usually used in legal contexts. However, due to the difficulty and complexity of the problem, the creation and enforcement of the legislation in a worldwide context was mostly unsuccessful. As such this approach was not fruitful and more direct approaches were used to combat spam emails.

In the works of [14], [15] changes to the transport layer and network layer of the internet. In particular [14] proposes a hardware architecture for a naive Bayes content classification unit for a high-throughput spam detection computation on the transportation layer to detect and clear spam mails as fast as possible and with minimum load on the whole network. In [15] a different email transfer protocol other than the currently used SMTP is recommended. This new protocol is named Differentiated Mail Transfer Protocol (DMTP). In this protocol, the receiver grants permission to different senders to send messages instead of the fundamentally sender-driven approach of SMTP. This especially has the potential to solve the root of the problem. However, due to many reasons such fundamental and large changes were not and could not be implemented, and as such more subtle changes were proposed.

In his thesis Shlomo Hershkop [16] arguess emails need a better search engine to extract information in them while a client is getting significant amounts of spam emails. For this approach, he proposes a Profile Email Toolkit (PET) which is a toolkit that applies data mining models to emails. PET could be used to transform the existing email client program from a simple reader to a more powerful analysis tool. In particular, this is achieved by using behavioral models to learn which messages are important to a particular [1] user and which are not. So this is a model which learns the preferences of each user and acts accordingly. However, this approach is not ideal for easy usage as such other models are developed.

In the works of [17], [18], [19], [20] filters on the application layer are proposed and discussed. These applications are basic machine learning algorithms based on content analysis. However, they also mention how the people who send spam emails have a monetary incentive to send spam and because of that they will always try to find a way to avoid these algorithms. As such these algorithms fail to detect ever-evolving spam mail due to their simple and rigid nature.

In [21] the writers present the first comprehensive measurement analysis of an actual anti-spam system using the challenge-response (CR) approach. CR systems shift the effort to protect the user's mailbox from the recipient to the sender of the messages. Also though these systems are often criticized due to their nature and these systems are believed to produce a lot of backscattered emails that further deteriorate the global Internet situation. This paper examines and experiments on a vast quantity of data gathered over a six-month period from over forty firms using a commercial challenge-response program with the aim of objectively looking at the CR approach for big firms.

From the cybersecurity point of view, interpreting emails that have the potential to be malicious is also very important. According to security reports [22], most people are psychologically analyzed according to their responses to malicious text or calls. This analysis later can be used in a lot of malware attacks, since the information about us also can affect our passwords and pins. Thus, blocking an email without even letting it be seen by the user is a crucial task.

As discussed in [23] HoneySpam is a system that uses honeypots technologies. The main concept is to reduce undesired traffic by combating spamming at the source rather than at the recipient. HoneySpam's characteristics include slowed email harvesting, poisoning of email databases with seemingly valid addresses, and improved spammer traceability through the use of fake open proxies and open relays.

As mentioned, machine learning systems use supervised and unsupervised machine learning algorithms to learn the behavior of spam and genuine emails and detect them automatically without the need for human interaction. [24], [25], [26], [27], [28], [29] Due to their highly adaptive nature and ability to find deeper underlying interactions these algorithms allow spam classifier algorithms to be one step ahead of the spam messages. In particaular for this research, with the developing and expanding application area of graph neural networks (GNN), they engaged in natural language processing frameworks [1].

One of the earliest graph models that applies GNNs to NLP problems is TextRank [30]. It represents text as graphs with edges that can be used to determine the relations between different words and word groups. In its paper, the authors show how it may be utilized in natural language processing applications. They offer two novel unsupervised approaches for keyword and phrase extraction and compare the results to published findings on recognized benchmarks.

According to the paper, TextRank works effectively because it uses information from the entire text, not just the local context of a text unit. TextRank uses graphs to identify relationships between entities in a text and implements recommendations. A text unit suggests other text units, and the suggestions' strength is depending on the units' importance. In the keyphrase extraction application, for example, co-occurring words are crucial, and the shared context identifies links between words in text. A sentence identifies essential phrases in a text by recommending another sentence that covers comparable themes. Sentences that are highly suggested by other sentences in the text will receive a higher grade.[30]

Moreover, TextRank's accuracy in both applications is comparable to other state-of-the-art algorithms. It is adaptable to various domains and languages since it does not require extensive linguistic expertise or domain knowledge [30].

Since these early graph models, there has been a growing interest in applying and creating multiple GNN variants with significant success in many NLP tasks like text categorization, semantic role labeling and translation software.

As an application in semantic role labeling, [31] utilized GNN’s to over-perform current methods. Their sentence encoders employ GCNs over syntactic dependency trees to provide latent feature representations of words in sentences. They find that stacking GCN and LSTM layers improves performance over an already advanced LSTM SRL model, resulting in very accurate results on the standard benchmark for both Chinese and English. Another work addressed this issue by proposing a lexicon-based graph neural network (LGN) with global semantics, in which lexical knowledge is employed to connect letters in order to capture local composition, while a global intermediate node may capture global word semantics and long-range reliance [32]. Word ambiguities can be efficiently addressed via numerous graph-based interactions among letters, possible words, and whole-sentence semantics. The graph is built with lexicons, which give word-level information. Based on the global sentence semantics, the LGN allows interactions between different sentence compositions and may capture non-sequential connections between letters. Experiments on NER datasets reveal that the proposed model outperforms competing baseline models significantly.

Convolutional neural networks (CNN) have feature extraction capability so that it is not necessary to give the features to the network manually. For email features, that makes the classification task a lot easier and accurate. [5] Conventional CNNs mostly have an embedding vector to extract the features, so before compiling the network it is necessary to give the embedded numerical vector representation of emails to extract the features. However, by using graph neural networks, instead of creating embedded numerical vectors from emails, the content of the email can be projected onto a graph and this graph can be classified using the graph neural network [33].

Projecting the content of an email to a graph is based on the analogy between semantic relations and graph connections. As a text can contain many structural relations between different words and word groups, graph representations are quite successful for text classification and analysis. There are different methodologies for this. Graph embedding methods are one of those that can simplify graph structure of the data and overcome node and edge classification tasks on graph neural networks [34], [35] however while capturing more complex structural relations in the graph matters, they may not perform that well. Graph convolutional networks succeed on this task as they are able to extract hierarchical and relational structures of the graph representations and solve the underlying meanings in a text by interpreting those relations [36].

3. Methods:

3.1 Graph Neural Networks:

Graph neural networks use the message passing algorithm between two maps. In each node map, nodes represent words or word groups that change their size according to n-gram size used in the training. In our case, in all the training sessions, the n-gram size was 3. These nodes are connected to others according to semantic relations, most of these connections are because of nodes being collocation candidates or occuring side-by-side. The goal here for the model is to be able to learn characteristics of a spam mail by observing the numerous occurrences of some certain words and word groups among these mails. By choosing n-gram size as 3, our purpose was to keep the model simple and make the training more time-efficient.

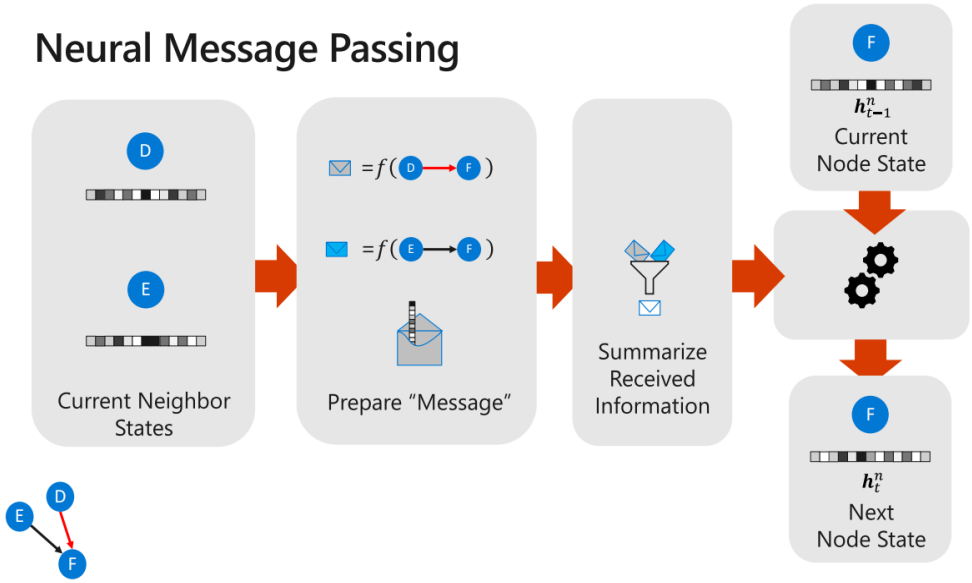

Figure 1: Message Passing Algorithm [37]

In the message passing algorithm, a node takes the information from other nodes connected to it, and updates itself on the next layer. Every node does that, hence according to the information from connected nodes, every node is updated in each layer. By taking time t as the layer number, the updating process can be represented layer-by-layer as seen below:

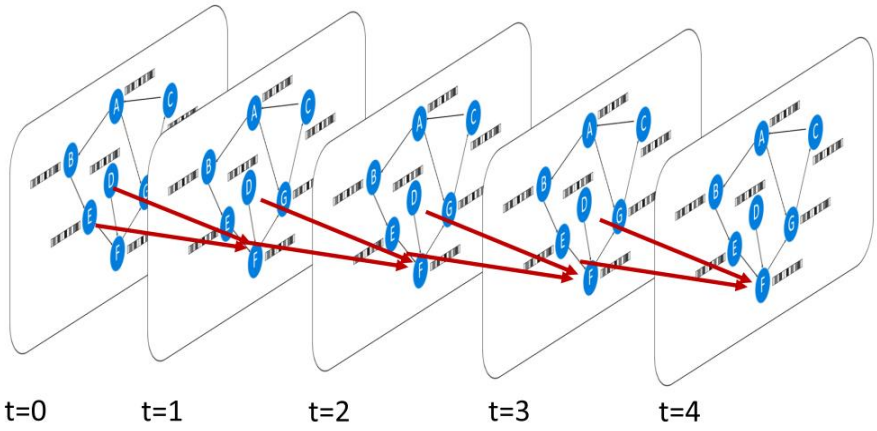

Figure 2: Updating Principle of Graph Neural Network [37]

In order to conserve location information of these nodes, applying convolution to the mapping information is quite useful. By using the layer by layer representation, the network involves a number of channels and gives output according to the resulting value. This provides us with gathering more information from multiple layers and the ability to combine them at the output.

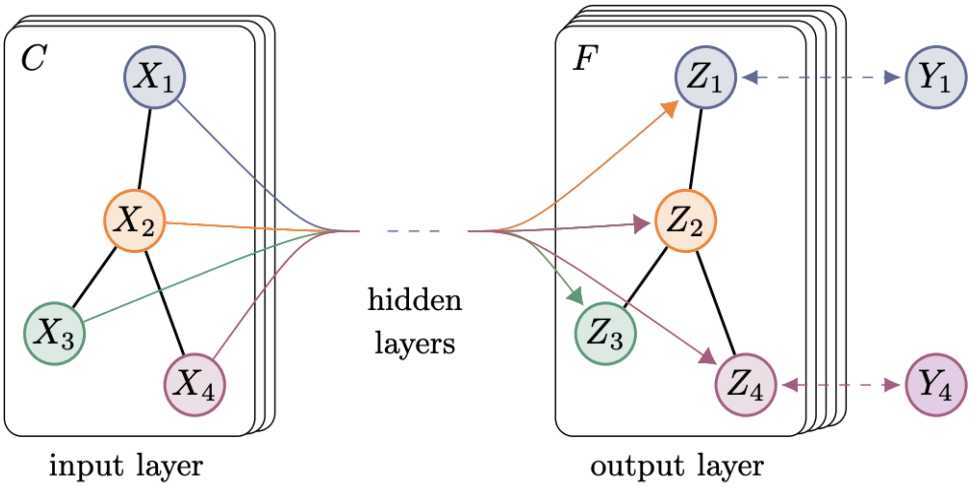

Figure 3: Layers of Graph Convolutional Network with Sample Input and Output [38]

3.2 Our Model:

Our model is quite similar to the model in [2]. In the model, each word is represented by a vector. The vector representations come from the Word2Vec and this will be further discussed later. The model builds a new graph

for each email. It connects each word node to three before and three after nodes, so each node is mutually connected to three to six other nodes based on its position in the email. It trains or predicts based on this graph and then decomposes the graph and moves on to the next mail. Thanks to this approach both the model consumes less memory and it is better at online training as updating parameters is less problematic compareyd to traditional GNN methods.

Additionally, our model creates a dictionary of all the used words in both training end tests and matches each word with an integer. Then it represents all emails as a list of integers. This is important as long strings take up a lot of memory so instead of them using an integer list is a lot faster. This way the most time consuming part of the algorithm which is moving strings from CPU to the GPU is cut short.

3.2.2 Preprocessing:

For the preprocessing we take the emails body only and discard the metadata part. Then we remove all the non-alpha characters. Then we lower case the data. We specifically don't remove the stop words as distances between content words could be important and removing stop words would mess up the distances. Then we tokenize the words. We specifically did not lemetize the words as different formats of a word carry different meaning and Word2Vec has different versions of the words so it captures this extraming without adding complexity to the model. So turning all the words to their root would just cause needless loss of information.

Something interesting to note is we do not remove numbers we just replace them with the word “the” as the is a word with next to no meaning and it fills up it allows us to preserve the distance relation of words without creating problems for us.

3.2.2 Model Specifications:

For the vector representation of the words the 200 dimensional version of word2vec which is trained on data from twitter. More specifically the word2vec is trained using two billion tweets which in total has 27 billion tokens and these tokens form a vocabulary of 1.2 million words. We delebirtly choose the vesion trained with tweets as emails and tweets are conceptialy are the most similar option.

In this model we use a non-spectral method named message passing mechanism that is used for convolution. The message passing mechanism collects information from the connected nodes and updates the nodes value based on the information collected from the adjacent nodes and the nodes original value. Specifically, we define the message passing mechanism as:

$$\boldsymbol{M_n} = \max_{a \in N_n^p} e_{an} \boldsymbol{r_a}$$
$$\boldsymbol{r'_n} = (1 - \eta_n)\boldsymbol{M_n} + \eta_n \boldsymbol{r_n}$$

Where $\mathbf{M_n}$ is the message node n received from its neighbors, e_{an} represents the edge weight from node a to node n and it is updated during the training η_n represents the learning rate of node n and it is also updated during the training. And the max operation is dimension based max operation where the max of each dimension is taken.

After the nodes are updated a sufficient number of times the decision rule is simply summing up the vector representation of the words with a bias term. So, we only add something close to a single layer of neurons at the end of our graph. Specifically, the final layer is defined as:

$$y_i = softmax\left(Relu\left(\boldsymbol{W}\sum_{n \in N_i} \boldsymbol{r'_n} + \boldsymbol{b}\right)\right)$$

Where **W** and **b** are trained during the training. It is crucial that we sum the vectors as it allows the model to be independent of the lengths of the mail. Without it or another method that does similar work, we can't train a model as each input would require a different number of parameters.

3.3 Used Data Sets:

The Enron-Spam dataset was obtained by the Federal Energy Regulatory while investigating the collapse of Enron. It contains approximately 500,000 emails generated by the employees of Enron.

The Kaggle data set is the private hand labeled emails of the author of the data made for the purpose of creating a challenge and it has 1000 mails in it.

Both data sets only contain the body of the email with the emails tag of spam-ham.

Additionally, the Enron dataset is collected between 2000 and 2006 while the Kaggle dataset is collected from 2020 as such they are quite different. Hence, comparing the algorithms performances on these datasets has a lot of potential for giving us an insight.

4. Results:

The statistical significance of the results is computed using Hoefflin's inequality. We make the null assumption of the algorithms randomly assigning spam and ham tags to the mails which would give an accuracy of 50% and compare that with the observed accuracy. So, the probability of observing the results we have observed for a randomly tag assigning algorithm is:

$$p \leq exp\left(\frac{-2(accuracy - 0.5)^2}{testset}\right)$$

Results of ENRON Dataset:

Check appendix, results section for figure 4 and figure 5 with explanations.

The test dataset performance for the ENRON dataset was as follows: 2140 out of 2176 predictions were correct, the test accuracy is measured as 0.9834559. The probability of acquiring these results for a random classifier is 0.0002148.

Results of Kaggle Dataset:

Check appendix, results section for figure 6 and figure 7 with explanations.

The test dataset performance for the Kaggle dataset was as follows: 186 out of 192 predictions were correct, the test accuracy is measured as 0.96875. The probability of acquiring these results for a random classifier is 0.002286.

5. Discussion and Conclusion:

GraphCNNs have been shown to be beneficial and effective on spam e-mail classification task in our experiments and research. We also observed that GCN models use less data and are more robust to hyper parameter changes. Although there is a significant disadvantage of GCN's, which is the lack of sequence information, since word ordering is not that critical for word classification, GCN's were quite successful in our experiments.

We presented a Graph Neural Network based approach for email categorization in this report. It transforms the email classification problem into a graph classification problem, after which it uses the GCN model to categorize the emails. We put our strategy to the test on a variety of public datasets. In terms of spam categorization, our results indicated that we outperformed a current state-of-the-art deep learning-based technique which is a semantic graph neural network that has been presented recently [33]. We believe that the superiority of our model compared to the one in [33] simply relies on the fact that we used a much simpler structure. Since the datasets we used were indeed simple datasets in which the spam e-mails were highly evident from their structures and specific words, it is reasonable to claim that a simpler model in terms of its structure which can generalize better can yield more accurate results than a highly complex model with a lot of learnable parameters. Finally, to improve the accuracy of our method in the future, we can use other preprocessing techniques such as word disambiguation and we can extend the model to be also effective in other classification problems.

References:

[1] Yao, L.; Mao, C.; Luo, Y. Graph Convolutional Networks for Text Classification. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 July–1 February 2019; pp. 7370–7377.

[2] Huang, L.; Ma, D.; Li, S.; Zhang, X.; Wang, H. Text Level Graph Neural Network for Text Classification. In Proceedings of the Empirical Methods in Natural Language Processing, Hong Kong, China, 3–7 November 2019; pp. 3442–3448.

[3] Zhang, Y.; Liu, Q.; Song, L. Sentence-State LSTM for Text Representation. In Proceedings of the Meeting of the Association for Computational Linguistics, Melbourne, Australia, 15–20 July 2018; pp. 317–327.

[4] A. Harisinghaney, A. Dixit, S. Gupta, and A. Arora, “Text and image based spam email classification using KNN, Naïve Bayes and Reverse DBSCAN algorithm,” 2014 International Conference on Reliability Optimization and Information Technology (ICROIT), Feb. 2014, doi: 10.1109/icroit.2014.6798302.

[5]A. Sharaff and H. Gupta, “Extra-Tree Classifier with Metaheuristics Approach for Email Classification,” Advances in Intelligent Systems and Computing, pp. 189–197, 2019, doi: 10.1007/978-981-13-6861-5_17.

[6] N. Bouguila and O. Amayri, “A discrete mixture-based kernel for SVMs: Application to spam and image categorization,” Information Processing & Management, vol. 45, no. 6, pp. 631–642, Nov. 2009, doi: 10.1016/j.ipm.2009.05.005.

[7] A. Derhab, A. Aldweesh, A. Z. Emam, and F. A. Khan, “Intrusion Detection System for Internet of Things Based on Temporal Convolution Neural Network and Efficient Feature Engineering,” Wireless Communications and Mobile Computing, vol. 2020, pp. 1–16, Dec. 2020, doi: 10.1155/2020/6689134.

[8] "The Enron-Spam datasets," 19 June 2006. [Online]. Available: http://www2.aueb.gr/users/ion/data/enron-spam/readme.txt. [Accessed 23 February 2022].

[9] A. Miglani, "E-Mail classification NLP," 2021. [Online]. Available: https://www.kaggle.com/datatattle/email-classification-nlp. [Accessed 23 February 2022].

[10] "SMS spam classification," 2016. [Online]. Available: https://www.kaggle.com/c/smsspam-classification/overview. [Accessed 23 February 2022].

[11] Kipf, T. and Welling, M., 2022. Semi-Supervised Classification with Graph Convolutional Networks. [online] arXiv.org. Available at: [Accessed 6 March 2022].

[12] N. Lugaresi, European union vs. spam: A legal response., in: CEAS, 2004.

[13] E. Moustakas, C. Ranganathan, P. Duquenoy, Combating spam through legislation: a comparative analysis of us and european approaches., in: CEAS, 2005.

[14] M. N. Marsono, Towards improving e-mail content classification for spam control: architecture, abstraction, and strategies, Ph.D. thesis (2007).

[15] Z. Duan, Y. Dong, K. Gopalan, Dmtp: Controlling spam through message delivery differentiation, Computer Networks 51 (10) (2007) 2616–2630.

[16] S. Hershkop, S. J. Stolfo, Behavior-based email analysis with application to spam detection, Columbia University, 2006.

[17] E. P. Sanz, J. M. G. Hidalgo, J. C. C. Pérez, Email spam filtering, Advances in computers 74 (2008) 45–114.

[18] S. Heron, Technologies for spam detection, Network Security 2009 (1) (2009) 11–15.

[19] P. Resnick, Internet message format (rfc 2822), Network Working Group, IETF (2001).

[20] G. V. Cormack, et al., Email spam filtering: A systematic review, Foundations and Trends R in Information Retrieval 1 (4) (2008) 335–455.

[21] J. Isacenkova, D. Balzarotti, Measurement and evaluation of a real world deployment of a challenge-response spam filter, in: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, ACM, 2011, pp. 413–426. 6

[22] Security threat report: Smarter, shadier, stealthier malware (2014). URL https://nakedsecurity.sophos.com/2013/12/11/smarter-shadier-stealthier-security-threat-report -2014-helps-you-understand-the-enemy/

[23] M. Andreolini, A. Bulgarelli, M. Colajanni, F. Mazzoni, Honeyspam: Honeypots fighting spam at the source., SRUTI 5 (2005) 11–11.

[24] G. V. Cormack, et al., Email spam filtering: A systematic review, Foundations and Trends R in Information Retrieval 1 (4) (2008) 335–455.

[25] J. Carpinter, R. Hunt, Tightening the net: A review of current and next generation spam filtering tools, Computers & security 25 (8) (2006) 566–578.

[26] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, Emerging artificial intelligence applications in computer engineering 160 (2007) 3–24.

[27] F. Qian, A. Pathak, Y. C. Hu, Z. M. Mao, Y. Xie, A case for unsupervised-learning based spam filtering., in: SIGMETRICS, Vol. 10, 2010, pp. 367–368.

[28] K.-N. Tran, M. Alazab, R. Broadhurst, et al., Towards a feature rich model for predicting spam emails containing malicious attachments and urls (2014).

[29] A. Bhowmick, S. M. Hazarika, Machine learning for e-mail spam filtering: review, techniques and trends, arXiv preprint arXiv:1606.01042 (2016).

[30] Mihalcea, R.; Tarau, P. TextRank: Bringing Order into Text. In Proceedings of the Empirical Methods in Natural Language Processing, Barcelona, Spain, 25–26 July 2004; pp. 404–411.

[31] D. Marcheggiani, J. Bastings, and I. Titov, “Exploiting Semantics in Neural Machine Translation with Graph Convolutional Networks,” Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), 2018, doi: 10.18653/v1/n18-2078. 7

[32] T. Gui et al., “A Lexicon-Based Graph Neural Network for Chinese NER,” Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, doi: 10.18653/v1/d19-1096.

[33] W. Pan, J. Li, L. Gao, L. Yue, Y. Yang, L. Deng and C. Deng, "Semantic Graph Neural Network: A Conversion from Spam Email Classification to Graph Classification", Scientific Programming, vol. 2022, pp. 1-8, 2022 [Online]. Available: https://www.hindawi.com/journals/sp/2022/6737080/#abstract. [Accessed: 06- Mar- 2022]

[34] A. Grover and J. Leskovec, "Scalable feature learning for networks", ACM Conferences, 2016. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2939672.2939754?casa_token=pmZLmQp7fSUAAAAA:13E_P6jrlQWj7QkayH9CwExqNrtg2xUqRT_LTqL10WcerA5T7_u6RsaxMqPTR4wwS1sF XGWagNg. [Accessed: 06- Mar- 2022]

[35] B. Perozzi, R. Al-Rfou and S. Skiena, "DeepWalk: online learning of social representations", ACM Conferences, 2014. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2623330.2623732?casa_token=oKAyu9G8r0kAAAAA:W0r7L5jAoG9JMt1UPSU43qIWjurxXBbe6bxE2nk13Wbc9z6LXcy3pgV_ktwj4eCcOIAw5eB zF1I. [Accessed: 06- Mar- 2022]

[36] I. Varlamis, D. Michail, F. Glykou and P. Tsantilas, "A Survey on the Use of Graph Convolutional Networks for Combating Fake News", Future Internet, vol. 14, no. 3, p. 70, 8 2022 [Online]. Available: https://www.mdpi.com/1999-5903/14/3/70/htm. [Accessed: 06- Mar- 2022]

[37] M. Allamanis, “Model - Miltos Allamanis,” 2020gnn.pdf, 2020. [Online]. Available: https://miltos.allamanis.com/files/slides/2020gnn.pdf. [Accessed: 08-May-2022].

[38] "Papers with Code - GCN Explained", *Paperswithcode.com*, 2022. [Online]. Available: https://paperswithcode.com/method/gcn. [Accessed: 26- May- 2022]

Appendix:

Contributions:

Each member worked collaboratively and even if a member had more responsibility in a specific section other members checked and improved the section. Member's responsibility for each section is given below:

Conceptualization: Aras Fırat Ünal, Batu Arda Düzgün, Burak Susam; Introduction: Aras Fırat Ünal, Abstract: Batu Arda Düzgün, Literature review: Aras Fırat Ünal, Batu Arda Düzgün, Burak Susam ;Discussion and Conclusion:Burak Susam, Results: Aras Fırat Ünal;Methodology: Batu Arda Düzgün; Algorithmic Implementaton; Aras Fırat Ünal, Batu Arda Düzgün, Burak Susam.

Results:

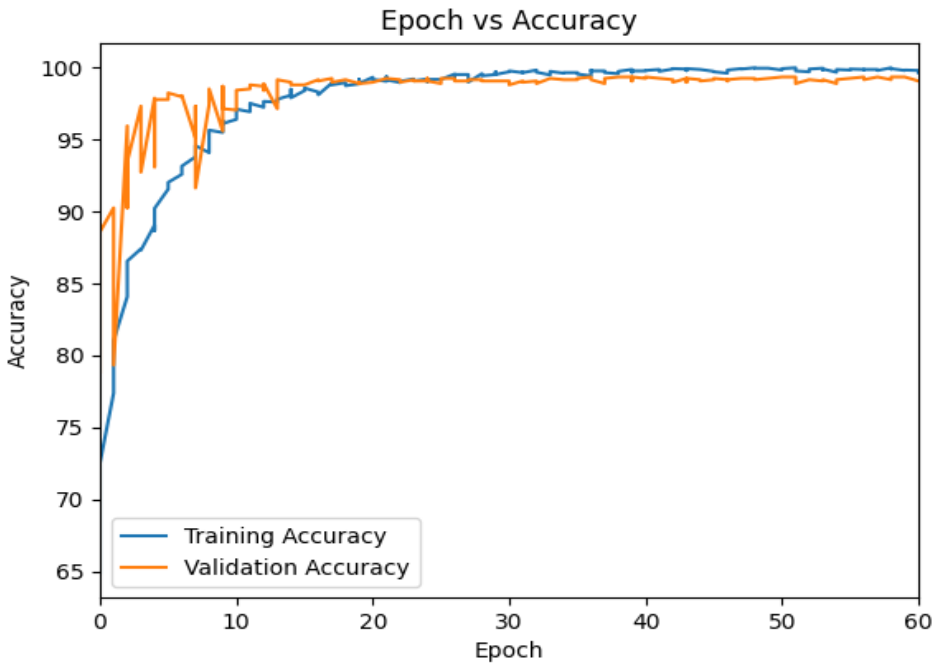

Figure 4: Plot showing accuracy of the model on both validation (orange) and training (blue) sets with respect to epochs. It is seen that after epoch 25-30, the accuracy performance of the network converges to a value around 99.2%. The similar trend that validation and training accuracies follow indicates the suitable complexity of the model in our case. Hence, no overfit or underfit occurred. In the early epochs which are lower than 10, some rapid changes occur in validation accuracy but after the model learns parameters, these changes disappear.

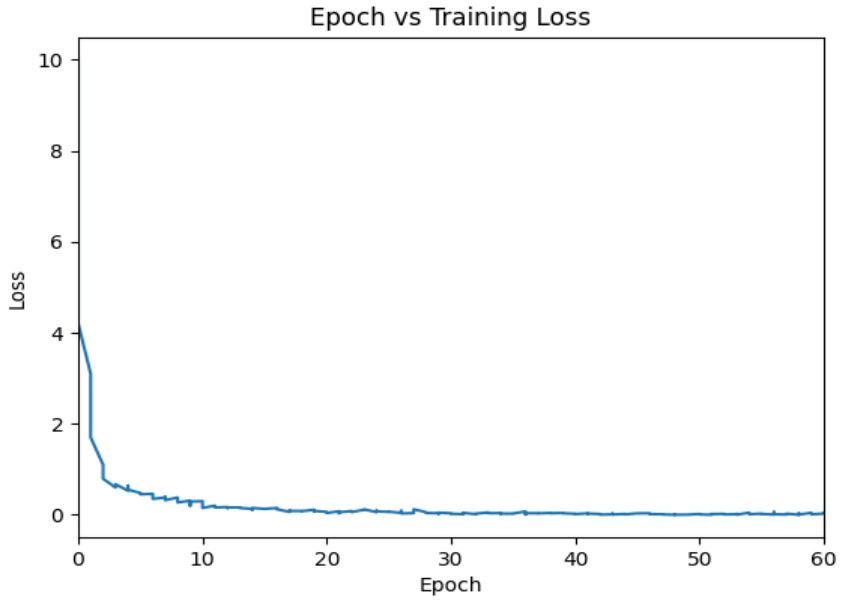

Figure 5: Plot showing training loss of the model with respect to epochs. This loss graph shows that the model is trained appropriately as the loss value decreases gradually over epochs. After epoch 30, loss value also converges to a value around 0.02, similarly to accuracy values as expected.

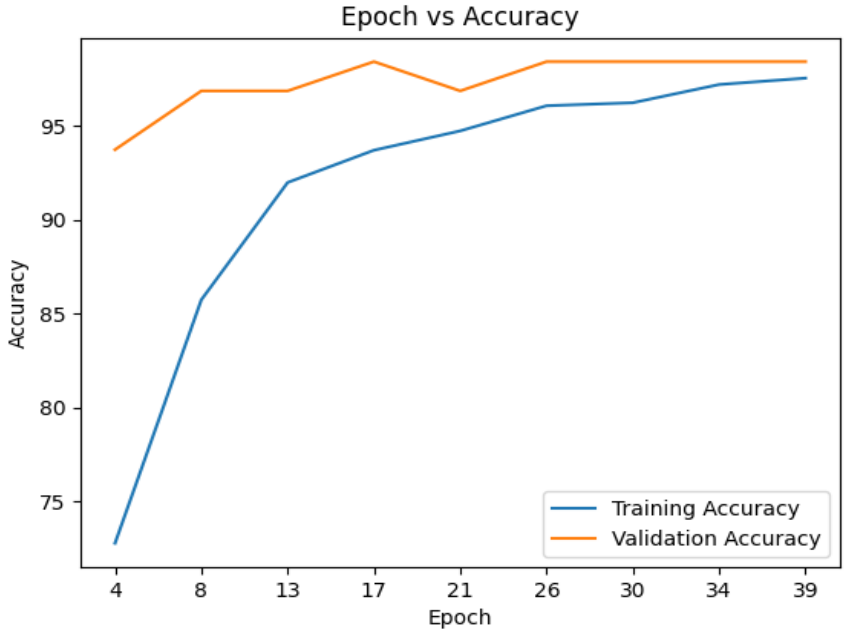

Figure 6: Plot showing accuracy of the model on both validation (orange) and training (blue) sets with respect to epochs. Since the Kaggle dataset is way smaller than our base dataset, it took significantly less time to train it. The difference between epochs was more significant and the number of epochs the training took for this level of accuracy at the end was 39. To comment on the complexity of the model for this case, no underfit or overfit occurred again as the training and validation accuracy is in a similar manner.

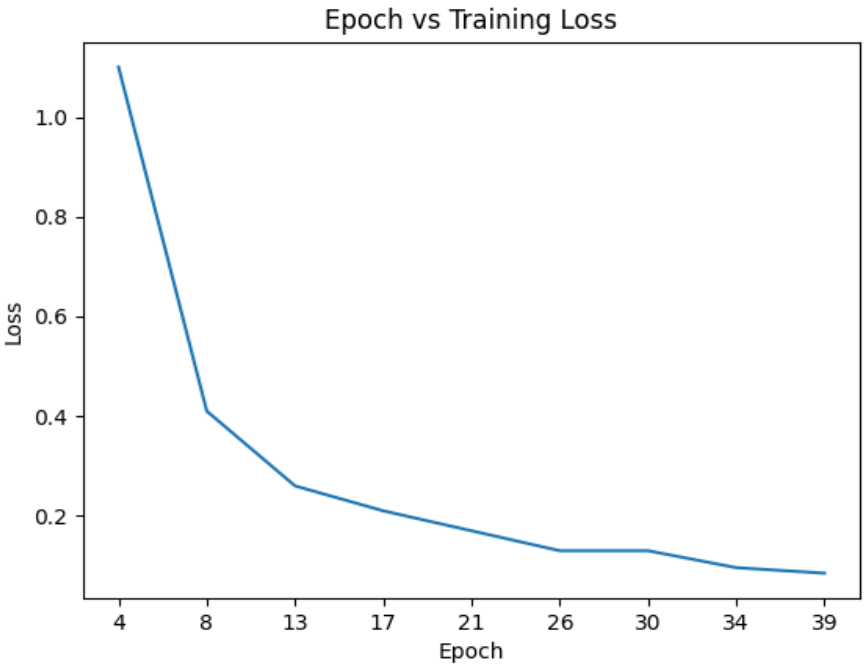

Figure 7: Plot showing training loss of the model with respect to epochs. This loss graph shows that the model is trained appropriately as the loss value decreases gradually over epochs. In the last epoch, we observe 0.085 as the final loss value.

Code:

main.py

```
import nltk

import time


import os

import string

import random as rd


import numpy as np


import custom_lemmatizer


from scipy.stats import binom
```

```
import math


import gensim.downloader


#nltk.download('wordnet')




stop_words = ["ourselves", "hers", "between", "yourself", "but", "again", "there", "about", "once", "during",
"out",
              "very", "having", "with", "they", "own", "an", "be", "some", "for", "do", "its", "yours", "such",
              "into", "of", "most", "itself", "other", "off", "is", "s", "am", "or", "who", "as", "from", "him",
              "each", "the", "themselves", "until", "below", "are", "we", "these", "your", "his", "through", "don",
              "nor", "me", "were", "her", "more", "himself", "this", "down", "should", "our", "their", "while",
              "above", "both", "up", "to", "ours", "had", "she", "all", "no", "when", "at", "any", "before", "them",
              "same", "and", "been", "have", "in", "will", "on", "does", "yourselves", "then", "that", "because",
              "over", "why", "so", "can", "did", "not", "now", "under", "he", "you", "herself", "has", "just",
              "where", "too", "only", "myself", "which", "those", "i", "after", "few", "whom", "t", "being", "if",
              "theirs", "my", "against", "a", "by", "doing", "it", "how", "further", "was", "here", "than"]


def Preproces(input_text, remove_stopwords):


    # removing the punctuation and apostrophe
    temp_text = input_text.translate(str.maketrans("'", " ", "‘’" + string.punctuation))


    # making it all lower case
    temp_text = temp_text.lower()


    if remove_stopwords:
```

```
        text_words = temp_text.split()
        NSW_text_words = [word for word in text_words if word.lower() not in stop_words]
        temp_text = ' '.join(NSW_text_words)

    return temp_text



print('The nltk version is {}.'.format(nltk.__version__))



ham_mails = []
spam_mails = []

ham_path = "ham"
spam_path = "spam"

print("\n\n*** reading the emials into list ***\n")

print("For Enron 1")
folder = "enron1"

start_time = time.time()
ham_adresses = os.listdir(f"{folder}\{ham_path}")
spam_adresses = os.listdir(f"{folder}\{spam_path}")

for address in ham_adresses:
    f = open(f"{folder}\{ham_path}\{address}", 'r', encoding='utf-8', errors='replace')
    ham_mails.append(f.read())
    f.close()
```

```
for address in spam_adresses:
    f = open(f"{folder}\{spam_path}\{address}", 'r', encoding='utf-8', errors='replace')
    spam_mails.append(f.read())
    f.close()


print("--- %s seconds taken reading the folder ---" % (time.time() - start_time))


print("\n\n*** Preprocesing the emails ***\n")
start_time = time.time()

for i in range(len(ham_mails)):
    ham_mails[i] = Preproces(ham_mails[i], False)

for i in range(len(spam_mails)):
    spam_mails[i] = Preproces(spam_mails[i], False)

print("--- %s seconds taken preprocessing the mails ---" % (time.time() - start_time))


print("\n\n*** Tokenizing the emails ***\n")
start_time = time.time()

ham_mail_tokens = []
spam_mail_tokens = []

for mail in ham_mails:
```

```
    temp_tokens = nltk.word_tokenize(mail)
    ham_mail_tokens.append(temp_tokens)

for mail in spam_mails:
    temp_tokens = nltk.word_tokenize(mail)
    spam_mail_tokens.append(temp_tokens)



print("--- %s seconds taken turning mails into tokens ---" % (time.time() - start_time))




print("\n\n*** creating vector representation of the tokens ***\n")
start_time = time.time()

ham_mail_token_vectors = []
spam_mail_token_vectors = []

# Download the "glove-twitter-50" embeddings
glove_vectors = gensim.downloader.load('glove-twitter-50')

print("--- %s seconds taken initilizing the word2vec ---" % (time.time() - start_time))
start_time = time.time()

for mail_index in range(len(ham_mail_tokens)):
    temp = []
    for token in ham_mail_tokens[mail_index]:
        if token in glove_vectors:
            temp.append(glove_vectors[token])
```

```
    ham_mail_token_vectors.append((ham_path, temp))

for mail_index in range(len(spam_mail_tokens)):
    temp = []
    for token in spam_mail_tokens[mail_index]:
        if token in glove_vectors:
            temp.append(glove_vectors[token])

    spam_mail_token_vectors.append((spam_path, temp))

print("--- %s seconds taken turning tokens into vectors ---" % (time.time() - start_time))

mergered_spam_ham_final = ham_mail_token_vectors + spam_mail_token_vectors
#rd.shuffle(mergered_spam_ham_final)

f = open("vector_mails.txt", 'w')

for mail in mergered_spam_ham_final:

    type = mail[0]
    vectors = mail[1]

    f.write(type + " ")

    for word_vector in vectors:
        for number in word_vector:
            f.write(str(number) + " ")
```

```
    f.write("\n")

f.close()

print("I am happy")
```

main_that_reads_data.py

```
import nltk
import time

import os
import string
import random as rd

import numpy as np

import custom_lemmatizer

from scipy.stats import binom

import math

start_time = time.time()

mergered_spam_ham_final = []


f = open("vector_mails.txt", 'r')

for mail in f.readlines():
```

```
    temp = mail[:-1]
    temp_list = temp.split()

    type = temp_list[0]

    temp_number_of_words_in_mail = (len(temp_list)-1)/50

    temp_word_vectors_of_mail = []

    for i in range(int(temp_number_of_words_in_mail)):

        temp_word_vector = temp_list[(i*50 + 1): (i*50 + 51)]

        temp_word_vectors_of_mail.append(temp_word_vector)


    mergered_spam_ham_final.append((type, np.array(temp_word_vectors_of_mail)))


f.close()

print("--- %s seconds taken to read the procesed data of the file ---" % (time.time() - start_time))



print("I am happy")
```

plotter.py

#This code takes console output as a txt file and converts it to a graph.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

f = open("graph_en.txt")
f = f.read()
lines = f.split("\n")
res_dict = {"Epoch": [], "Train Loss": [], "Train Acc": [], "Validation Acc": []}
lines.pop(-1)

for line in lines:
    res_dict["Epoch"].append(line[9:14].strip())
    res_dict['Train Loss'].append(float(line[41:47].strip()))
    res_dict['Train Acc'].append(float(line[60:66].strip()))
    res_dict['Validation Acc'].append(float(line[77:82].strip()))

res_df = pd.DataFrame.from_dict(res_dict)

plt.plot(res_df["Epoch"], res_df["Train Loss"])
plt.title("Epoch vs Training Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.xlim([0,60])
plt.xticks(np.arange(0, 61, 10))
plt.savefig("results/en/train_loss.png")
plt.clf()

plt.plot(res_df["Epoch"], res_df["Train Acc"])
```

```
plt.title("Epoch vs Training Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.xlim([0,60])
plt.xticks(np.arange(0, 61, 10))
plt.savefig("results/en/train_acc.png")
plt.clf()

plt.plot(res_df["Epoch"], res_df["Validation Acc"])
plt.title("Epoch vs Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.xlim([0,60])
plt.xticks(np.arange(0, 61, 10))
plt.savefig("results/en/validation_acc.png")
plt.clf()

plt.plot(res_df["Epoch"], res_df["Train Acc"])
plt.plot(res_df["Epoch"], res_df["Validation Acc"])
plt.title("Epoch vs Accuracy")
plt.legend(("Training Accuracy", "Validation Accuracy"))
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.xlim([0,60])
plt.xticks(np.arange(0, 61, 10))
plt.savefig("results/en/both_acc.png")
```

all_in_one.py

```
import torch
import numpy as np
```

```
from model import *
from pmi import *
import tqdm
import sys, random
import argparse
import dgl as d_graph
import time, datetime
import os
from data_loader import DataHelper
import pandas as pd


iter_num = 100
early_epoch_stop = 25



def trn_edg_mapping(vocabulary_length, cont, ngram):
    cnt = 1
    mapping = np.zeros(shape=(vocabulary_length, vocabulary_length), dtype=np.int32)
    for doc in cont:
        for i, source in enumerate(doc):
            for dest_id in range(max(0, i-ngram), min(length(doc), i+ngram+1)):
                dest = doc[dest_id]


                if mapping[source, dest] == 0:
                    mapping[source, dest] = cnt
                    cnt += 1


    for word in range(vocabulary_length):
        mapping[word, word] = cnt
```

```
        cnt += 1

    return cnt, mapping


def get_time_dif(start_time):
    """获取已使用时间"""
    end_time = time.time()
    time_dif = end_time - start_time
    return datetime.timedelta(seconds=int(round(time_dif)))


def dev(model, dataset):
    data_loader = DataHelper(dataset, mode='dev')

    total_pred = 0
    correct = 0
    iter = 0
    for cont, lbl, _ in data_loader.batch_iter(batch_size=64, num_epoch=1):
        iter += 1
        model.eval()

        logits = model(cont)
        pred = torch.argmax(logits, dim=1)

        correct_pred = torch.sum(pred == lbl)

        correct += correct_pred
        total_pred += length(cont)
```

```
    total_pred = float(total_pred)
    correct = correct.float()
    return torch.div(correct, total_pred)


def test(model_name, dataset):
    model = torch.load(os.path.join('.', model_name + '.pkl'))

    data_loader = DataHelper(dataset, mode='test')

    total_pred = 0
    correct = 0
    iter = 0
    for cont, lbl, _ in data_loader.batch_iter(batch_size=64, num_epoch=1):
        iter += 1
        model.eval()

        logits = model(cont)
        pred = torch.argmax(logits, dim=1)

        correct_pred = torch.sum(pred == lbl)

        correct += correct_pred
        total_pred += length(cont)

    total_pred = float(total_pred)
    correct = correct.float()
    print("Correct: ", correct, "\nTotal Predictions: ", total_pred)
```

```
    return torch.div(correct, total_pred).to('cpu')



def train(ngram, name, bar, drop_out, dataset, is_cuda=False, trn_edg=True):
    print('load data helper.')
    data_loader = DataHelper(dataset, mode='train')
    if os.path.exists(os.path.join('.', name+'.pkl')) and name != 'temp_model':
        print('load model from file.')
        model = torch.load(os.path.join('.', name+'.pkl'))
    else:
        print('new model.')
        if name == 'temp_model':
            name = 'temp_model_%s' % dataset
        trn_edg_weights, trn_edg_mappings, cnt = cal_PMI(dataset=dataset)

        model = Model(class_num=length(data_loader.lbls_str), hidden_size_node=200,
                      vocabulary=data_loader.vocabulary, n_gram=ngram, drop_out=drop_out,
trn_edg_matrix=trn_edg_mappings, trn_edg_num=cnt,
                      trainable_trn_edg=trn_edg, pmi=trn_edg_weights, cuda=is_cuda)

    print(model)
    if is_cuda:
        print('cuda')
        model.cuda()
    loss_func = torch.nn.CrossEntropyLoss()

    optim = torch.optim.Adam(model.parameters(), weight_decay=1e-6)

    iter = 0
    best_acc = 0.0
```

```
last_best_epoch = 0

start_time = time.time()

total_loss = 0.0

total_correct = 0

total = 0

res_dict = {"Epoch": [], "Train Loss": [], "Train Acc": [], "Validation Acc": []}

for cont, lbl, epoch in data_loader.batch_iter(batch_size=32, num_epoch=200):

    improved = ''

    model.train()


    logits = model(cont)

    # import pdb

    # pdb.set_trace()

    logits = logits.to(model(cont))

    loss = loss_func(logits, lbl)


    pred = torch.argmax(logits, dim=1)


    correct = torch.sum(pred == lbl)


    total_correct += correct

    total += length(lbl)


    total_loss += loss.item()


    optim.zero_grad()

    loss.backward()

    optim.step()
```

```
        iter += 1
        if iter % iter_num == 0:

            val_acc = dev(model, dataset=dataset)
            if val_acc > best_acc:
                best_acc = val_acc
                last_best = epoch

            if epoch - last_best>= early_epoch_stop:
                return name
            msg = 'Epoch: {0:>6} Iter: {1:>6}, Train Loss: {5:>7.2}, Train Acc: {6:>7.2%}' \
                + 'Val Acc: {2:>7.2%}, Time: {3}{4}' \

            print(msg.format(epoch, iter, val_acc, get_time_dif(start_time), total_loss/ iter_num,
                             float(total_correct) / float(total)))

            res_dict["Epoch"].append(epoch)
            res_dict['Train Acc'].append(float(total_correct) / float(total))
            res_dict['Train Loss'].append(total_loss/ iter_num)
            res_dict['Validation Acc'].append(val_acc)
            total_loss = 0.0
            total_correct = 0
            total = 0

    res_df = pd.DataFrame.from_dict(res_dict)
    pd.DataFrame.to_csv(res_df, "results.csv")
    return name

def wrd_ev():
```

```
    print('load model from file.')
    data_loader = DataHelper('r8')
    trn_edg_num, trn_edg_matrix = trn_edg_mapping(length(data_loader.vocabulary), data_loader.cont, 1)
    model = torch.load(os.path.join('wrd_ev_1.pkl'))

    trn_edg_weights = model.seq_edge_w.weight.to('cpu').detach().numpy()

    core_word = 'billion'
    core_index = data_loader.vocabulary.index(core_word)

    results = {}
    for i in range(length(data_loader.vocabulary)):
        word = data_loader.vocabulary[i]
        n_word = trn_edg_matrix[i, core_index]

        if n_word != 0:
            results[word] = trn_edg_weights[n_word][0]

    sort_results = sorted(results.items(), key=lambda d: d[1])

    print(sort_results)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--ngram', required=False, type=int, default=4, help='ngram number')
    parser.add_argument('--name', required=False, type=str, default='example_model')
    parser.add_argument('--dropout', required=False, type=float, default=0.5)
    parser.add_argument('--dataset', required=True, type=str, help='dataset')
```

```
parser.add_argument('--trn_edg', required=False, type=int, default=1

parser.add_argument('--rand', required=False, type=int, default=7)


args = parser.parse_args()


print('ngram: %d' % args.ngram)

print('project_name: %s' % args.name)

print('dataset: %s' % args.dataset)

print('trainable_trn_edg: %s' % args.trn_edg)

rnd_seed = args.rand



if args.trn_edg == 1:

    trn_edg = True

    print('trainable trn_edg')

else:

    trn_edg = False


model = train(args.ngram, args.name, args.dropout, dataset=args.dataset, is_cuda=False, trn_edg=trn_edg)

test_res = test(model, args.dataset).numpy()

print('test acc: ', test_res)
```