Contents

1. Project Scope 3

2. Data 3

2.1 Data Preprocessing 3

2.2. Data Exploration 4

2.3. Text Representation 9

2.4. Graph Representation 9

3. Modelling 10

3.1. Parameter Optimization 11

3.2. Benchmark 11

3.2. Final model 12

3.3. Model trials 19

4. Further Improvements 20

1. Project Scope

The goal of the project is to perform classification on a collection of data that represent web domains. Specifically, the purpose is to apply Machine Learning techniques in order to predict which class represents each domain in the most optimal way. This task can be classified in the problem categories of ***text classification*** combined with ***node classification***, since the data not only contain textual representation of the websites but we also obtain a graph of the domains, based on the links between them.

2. Data

The data provided for text classification consist of Greek domains. It consists of four main directories:

- ***Domains.zip****:* This file contains the textual content of the domains that are included in our project.
- ***Edgelist.txt****:* This file contains a large part of the Greek web graph stored as an edge-list. The nodes represent the domain names and edges represent the hyperlinks that can be found in the domain.
- ***Train.txt:*** This file contains the domain names that will constitute the train set of the classification task. These domains are already classified in one of the 9 classes, which represent the theme of the domain.
- ***Test.txt:*** This file contains the domain names that will constitute the test set of the classification task. Similarly, each of these domain names belongs to one of the 9 possible classes. There are the domains which the models will be evaluated on, on Kaggle.

2.1 Data Preprocessing

In order to clean up the text, we perform preprocessing suitable for natural language processing tasks. Specifically, we utilize the Greek language model from ***spaCy***. This model includes components necessary for NLP tasks such as tokenization, part-of-speech tagging, and lemmatization specific to the Greek language. Using a pre-trained model ensures that we have accurate and efficient tools for processing Greek text, leveraging spaCy's robust NLP capabilities. The steps included in the preprocessing are the following:

i. Firstly, we **truncate** the textual representations of the domains in an upper limit, not only for memory efficiency purposes but also to discard any noise in the data. The upper limit we choose to use is the average length of the text, which is **10,095 tokens**.
ii. Then, we remove **new line characters and URLs**, which are present in each text due to the nature of the data.
iii. Following, we **remove the accents** which are present due to the Greek language of the texts and we **convert** all characters to **lowercase**. This is performed for text normalization purposes but also to be in line with the requirements of the Embeddings' model we use later on.
iv. For each token, we apply **lemmatization** to reduce it to its base form and **filter** out **non-alphabetic tokens, stop-words, and words with less than three characters**.

The lemmatization helps in unifying different forms of a word and therefore helps in reducing the noise and in producing more compact text representations. The filtering focuses on meaningful words, since mainly small words in Greek do not have semantic value, and therefore reduces the noise and improves the computational efficiency.

This preprocessing pipeline as described above effectively prepares Greek text for the following classification tasks, by ensuring the domain data is consistent, simplified, and focused on meaningful content.

All the steps above are performed on the train and test domains, separately.

2.2. Data Exploration

Following on our analysis, we perform Data exploration to better understand the available data. To do so, we combine the information from the textual context of the domains and the information from the graph and the connectivity of the nodes.

Starting on our analysis, we study the class distribution of the train dataset. As we can see on the figure below, the classes are highly imbalanced. The Class 3 seems to have the highest percentage of the train samples, while the Classes 0, 6 and 7 have the lowest percentages.

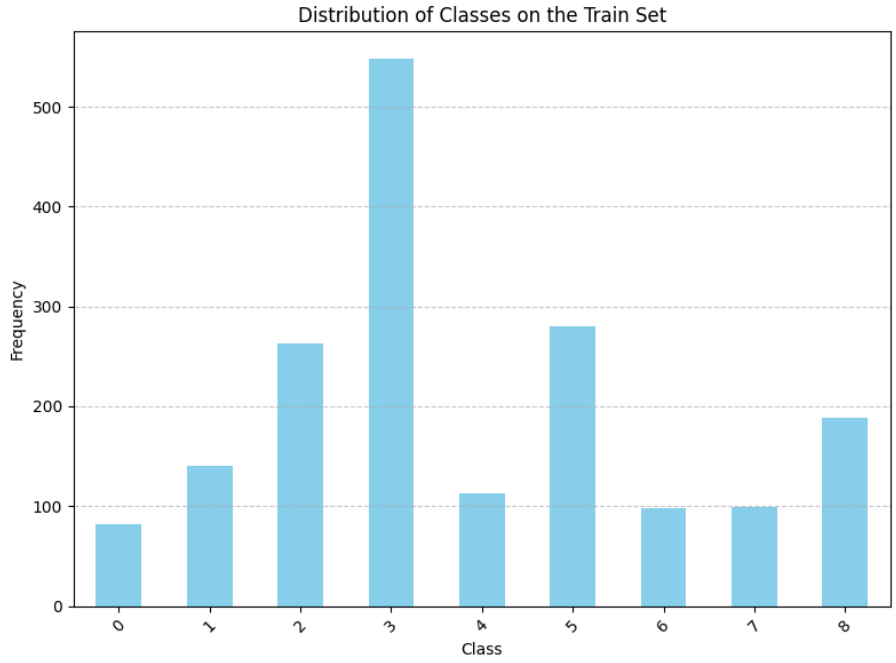


Image 1: Class distribution on the train dataset

In order to gain a better understanding on the classes and what each one of them represents, we analyze more the domains in each class. As we can see below, we print indicatively 10 domains of each class.

Indicative domains per each Class

	Domain_1	Domain_2	Domain_3	Domain_4	Domain_5
0	autocarnet.gr	bikerspoint.gr	mydirect.gr	f1fan.gr	asfalistra.gr
1	coachbasketball.gr	basketplus.gr	sportdepot.gr	velocitybikes.gr	cosmossport.gr
2	yourate.gr	theacropolismuseum.gr	rockandroll.gr	guestlist.gr	naftotopos.gr
3	athensgo.gr	kafeneio-gr.blogspot.gr	onalert.gr	mykosmos.gr	kriti24.gr
4	oaed.gr	education.gr	livepedia.gr	paideia-ergasia.gr	pi-schools.gr
5	queen.gr	aggeliorama.gr	kraxtis-gr.blogspot.gr	hostplus.gr	pixeldraw.gr
6	topgamos.gr	spitistalefka.gr	psistis.gr	flexystrom.gr	decofairy.gr
7	disaki.blogspot.gr	holiday.gr	yahotels.gr	gtp.gr	money-tourism.blogspot.gr
8	angelopouloshair.gr	sweetandbalance.gr	psychologos-vasilikiliafou.gr	healingeffect.blogspot.gr	psychotherapeia.net.gr

Domain_6	Domain_7	Domain_8	Domain_9	Domain_10
dim-tires.gr	mini.com.gr	citroen.gr	caranddriver.gr	moto-plus.gr
bianchi.gr	koe.org.gr	athlitiko.gr	redvoice.gr	sporfm.gr
rythmosweb.gr	the-walking-dead-greek-fanatics.gr	wehellas.gr	disney.gr	tsekouratoi.gr
tribune.gr	rthess.gr	verena.gr	rednotebook.gr	makeleio.gr
csap.gr	jobstoday.gr	paster.gr	unipi.gr	pliroforikiatschool.blogspot.gr
solygeia.blogspot.gr	specialone.gr	e-kalithea.gr	cyta.com.gr	aboutnet.gr
all4mama.gr	koykoycook.gr	livewall.gr	arttable.gr	epiplo-telioridis.gr
europe-hotels.gr	samothraki.gr	mega-holidays.gr	lefkada-ionio.gr	pamediakopes.gr
ladyslife.gr	diaitologia.blogspot.gr	anassa.gr	ygeianet.gov.gr	dhi.gr

Image 2: Ten random domains from each of the 9 classes split in two images for better appearance purposes.

We can already detect a theme for each class, but we also print the top words that appear most frequently in each class and have relatively large length, over 10 letters. In the below Image we can see indicatively some of them.

```
==========================================================================================
Class 0 - Top 20 words: ['συμπληρωστε', 'ερωτηματολογιο', 'αξιοπιστιος', 'αυτοκινητου', 'παραδοθηκε', 'κυ
==========================================================================================
Class 1 - Top 20 words: ['παιδαγωγικος', 'παιδαγωγικη', 'προερχεται', 'παιδαγωγος', 'ακριβολογωντα', 'εκπ
==========================================================================================
Class 2 - Top 20 words: ['κατεβαζοντα', 'χρειαζεται', 'προσαρμοστω', 'τηλεοπτικο', 'εκπληκτικος', 'διευθυ
==========================================================================================
Class 3 - Top 20 words: ['κοινοποιηση', 'συμπεριφορα', 'περιμενατε', 'καταλαβετε', 'υποστηριξει', 'κατανο
==========================================================================================
Class 4 - Top 20 words: ['αναβαθμιση', 'συστηματων', 'πληροφορικη', 'υποφακελος', 'τελευταιας', 'διαδικτυ
==========================================================================================
Class 5 - Top 20 words: ['φωτογραφια', 'τηλεοπτικη', 'φαντασμαγορικος', 'αντικειμενος', 'αντιστοιχα', 'πα
==========================================================================================
Class 6 - Top 20 words: ['παντοβασιλισσας', 'μοιραστειτε', 'περισσοτερα', 'επισκεπτες', 'επιθυμειτε', 'εγ
==========================================================================================
Class 7 - Top 20 words: ['φεβρουαριου', 'φεβρουαριος', 'αμερικανικος', 'αεροδρομιο', 'αναμνησεις', 'αναχω
==========================================================================================
Class 8 - Top 20 words: ['προστεθηκε', 'γλυκαιμικου', 'πανεπιστημιο', 'αυστραλιος', 'γλυκαιμικος', 'ερευν
==========================================================================================
```

Image 3: Indicative the top famous words in the texts per Class

Following, we focus more on the graph information in our data. First of all, the graph of all the domains consists of 65,208 nodes and 1,642,073 edges.

In order to gain valuable information for the nodes, we compute the in-degree, out-degree, and average neighbor degree for each domain node in the training set:

- **In-degree**: This metric represents the number of incoming edges to a node. Therefore, it can indicate how popular a domain can be, based on how many other domains point at it.

- **Out-degree**: This metric denotes the number of outgoing edges from a node. Therefore, nodes with high out-degree are possibly nodes that are very active in making connections and they could be seen hubs that frequently point to other nodes.

- **Average neighbor degree**: This measure indicates the average degree of a node's neighbors. For a node v, the average neighbor degree can be computed by averaging the degrees of all nodes that are adjacent to v. So, nodes with high average neighbor degree means that they are connected to other nodes that themselves have a high degree. This can indicate that the node is part of a well-connected or influential neighborhood within the graph.

After computing this information, we study the nodes with the highest in-degree, out-degree and average neighbors' degree per Class.

The domains per class with the highest in-degrees:

	Domain	Class	in_degree_x
1653	car.gr	0	515
282	gazzetta.gr	1	2017
617	sansimera.gr	2	3069
1785	tovima.gr	3	6340
619	uoa.gr	4	3693
768	google.gr	5	7885
248	mothersblog.gr	6	694
1007	oasa.gr	7	823
310	iatronet.gr	8	1491

Image 4: The domains per Class that have the highest in –degrees.

The domains per class with the highest out-degrees:

	Domain	Class	out_degree_x
1422	moto.gr	0	442
490	podilates.gr	1	1976
1486	slang.gr	2	1475
774	in2life.gr	3	3435
1368	apn.gr	3	3435
261	auth.gr	4	1334
482	freestuff.gr	5	3519
1630	hamomilaki.blogspot.gr	6	2102
528	arttravel.gr	7	848
570	iator.gr	8	564

Image 5: *The domains per Class that have the highest out –degrees*

The domains per class with the highest average neighbors degrees:

	Domain	Class	avg_neighbor_degree_x
1528	kokkinakis-service.gr	0	477.428571
911	sportcyclades.gr	1	278.432432
1686	sfgame.gr	2	1760.400000
1281	kalamatajournal.gr	3	1248.800000
1676	epimorphosi.gr	4	790.454545
1147	pitsirikos.gr	5	957.000000
1317	koykoycook.blogspot.gr	6	692.200000
508	rexhotelthessaloniki.gr	7	2159.000000
627	logosglyfadas.gr	8	2159.000000
1286	physio-activity.gr	8	2159.000000
1410	palaioxorinos-orl.gr	8	2159.000000

Image 6: *The domains per Class that have the highest average neighbors' degrees.*

As we can see from the above, and especially from in-degrees and out-degrees results, there seem to be a clear theme for each Class. If we combine all of the above insights we can conclude the following:

Class	*Theme*
0	*Driving/Cars*
1	*Sports*
2	*Art*
3	*News*
4	*Education*
5	*Weather Forecasting, Customer Reviews and Online Transactions*
6	*Housing and Home ware*
7	*Traveling*
8	*Health*

Next in our analysis, we study the communities that are formed on the train domains in each of the 9 Classes. To do so we utilize the Louvain method, which can provide valuable insights into the structural organization and cohesiveness of the nodes within each class and we calculate the number of communities per Class.

```
Class 0 has 10 communities.
Class 1 has 11 communities.
Class 2 has 10 communities.
Class 3 has 9 communities.
Class 4 has 9 communities.
Class 5 has 10 communities.
Class 6 has 8 communities.
Class 7 has 7 communities.
Class 8 has 10 communities.
```

Image 7: Number of Communities in the train domains, per Class

The number of communities detected in each class indicates how the nodes within each class are grouped based on their connections. In our data, the numbers range from 7 to 11

communities per class, however there is no huge difference between the classes. Some remarks on that:

- **Classes with Higher Number of Communities (10-11)**: Classes 0, 1, 2, 5, and 8 have more communities than the rest. This might suggests that these classes have a more complex structure with multiple distinct subgroups. These classes may have more diverse or varied content, leading to the formation of more subgroups within the network, which could represent different themes, or areas of interest.
- **Classes with Fewer Communities (7-9)**: Classes 3, 4, 6, and 7 have fewer communities. This indicates a more cohesive structure with fewer groups. These classes might have more homogenous content or stronger interconnections among the nodes, resulting in fewer, more significant clusters.

Finally, before we continue with the rest components of the model development, we print some informative statistics on the train and test dataset, but also per each of the Classes.

```
Size of train set: 1812
Size of test set: 605
Avg document length (train): 3001
Avg document length (test): 2939
Vocabulary size: 195946
Out of vocabulary words (test): 30428
=====================================
Average Document Length per Class:
Class 0 - Avg doc length 2079
Class 1 - Avg doc length 3256
Class 2 - Avg doc length 3062
Class 3 - Avg doc length 3813
Class 4 - Avg doc length 2503
Class 5 - Avg doc length 2408
Class 6 - Avg doc length 2715
Class 7 - Avg doc length 1992
Class 8 - Avg doc length 2616
```

Image 8: Statistics on the textual representation of the domains

The vocabulary size is quite large, reflecting a rich and diverse set of words used across the textual content of the domains. However, there are 30,428 out-of-vocabulary (OOV) words in the test set, which is significant. This could imply that there are many words in the test set that the model has not seen during training, potentially affecting the model's ability later on to accurately interpret test data.

Generally, the average lengths of the texts per class seem similar. However, **Class 3** has the highest average document length (3,813 tokens), indicating that documents in this class tend to be longer and possibly more detailed. As a reminder, this Class seems to contain domains from the “News” domain, so this can explain this fact.

2.3. Text Representation

After the data preprocessing & exploration, we continue with creating a valid representation of the text data, in order to use them in the classification models. There are various choices on this area (e.g., tf-idf representation, embeddings using pretrained Greek model from fast-text), but the most helpful representation was achieved by utilizing **pre – trained models based on BERT** and extracting the embeddings of the words in our text.

As we know, **BERT** provides embeddings with contextual information, allowing them to capture the meaning of words based on their surrounding context, which often leads to more accurate representations for downstream tasks. Furthermore, due to the limited availability of high-quality models for Greek text, using BERT embeddings becomes crucial as it outperforms other models (e.g. fast-text that was also used, did not result in good performance).

Specifically, since the textual representation of the domains is in the Greek language, we used a **Sentence Transformer** based on the **Greek media Bert model**[1]. This model was trained on a custom dataset containing triplets from the combined Greek "internet", "social-media" and "press" domains. The dataset was created by sampling triplets of sentences from the same domain, where the first two sentences are more similar than the third one. Training objective was to maximize the similarity between the first two sentences and minimize the similarity between the first and the third sentence.

By utilizing this pre-trained model, each sentence is mapped to a **768 - dimensional** dense vector space, which will be used as input to our classification models later.

It should be noted, that besides this transformer, we tested another Bert based model, which is also trained in Greek texts[2]. Specifically, we obtained the embeddings of the tokens of each sentence and produced the sentence embedding by averaging the tokens' embeddings. However, by comparing the performance of these embeddings with the embeddings produced from the Sentence Tokenizer on our baseline model (Simple Logistic Regression model, described below), we concluded that the Sentence Tokenizer embeddings produce more accurate representations. This discrepancy between those two might be due to the way these models were trained or the data that were used, since the Sentence Tokenizer mentions that it was trained on social media data and press domains, which might be more similar to our data.

Therefore, to conclude each sentence/record is of shape **(768,)** and therefore our initial training set consists of 1,812 sentences of size 768.

2.4. Graph Representation

Up to here, we have managed to represent the textual information of the domains; however we also need to capture information from the graph, in order to make the classification more accurate.

[1] https://huggingface.co/dimitriz/st-greek-media-bert-base-uncased
[2] https://huggingface.co/nlpaueb/bert-base-greek-uncased-v1

Specifically, we capture the connectivity of the nodes – domains in the graph based on the following methodology:

- Firstly, we generate **random walks** (20 random walks with max depth of 40 nodes) for nodes in the graph. Each walk starts from a domain node and randomly traverses neighboring domains, capturing local domain relationships.
- Then, we represent each random walk sequence as a "text" document, where nodes visited in the random walk correspond to "words" in the text. We utilize and train **Word2Vec** model to learn embeddings for each node based on these sequences, capturing the structural information of the graph and contextual relationships between domains. This word2Vec model is a skip-gram model which basically learns to predict the neighboring domains given a domain, and we use a window size of 10, meaning the model considers 10 neighboring domains on each side of the domain during training.

Therefore, from the procedure above, each domain node in the graph will be represented by a dense vector of size **(128,)** which is our chosen embedding size in the Word2Vec model, capturing the structural information and relationships with other domains in the graph.

In addition to using random walks for generating node embeddings, we also incorporate structural features for each domain node

Specifically, we utilize the **in-degree**, **out-degree**, and **average neighbor degree**. These features are also explained in the Data Exploration analysis, however in general they provide information about the connectivity of nodes, as the in-degree represents the node popularity within the network, the out-degree represents the node's activity and finally the average neighbor degree provides insights into the connectivity of a node's neighbors, helping to understand the local network density around the node.

We also extracted an additional feature **(1,)** by converting the graph to an undirected graph and partitioning the nodes into communities based on their connectivity patterns using the Louvain algorithm.

Combining these four features with the learned embeddings from Word2Vec (deep walks), we get node representations of size **(132,)**, which will later be used in the modeling process.

3. Modelling

In this section, we describe the developed final model and some trial models that achieved a satisfactory score. We aimed to leverage both textual information and graph structural features in order to achieve high classification accuracy. To achieve this, we developed the following model architecture that effectively integrates both the text and the graph data, enabling comprehensive analysis and capturing the interplay between textual content and network structure for improved performance.

3.1. Parameter Optimization

In order to find the optimal hyper – parameters that could be used in the models developed and mentioned below, we opted to perform tuning using the ***Optuna*** framework[3]. Optuna is an open-source optimization library, which searches the optimal solution by trial and error in the search space that is defined by us (by defining the lower value, the upper value and the step size (wherever needed) of each hyper-parameter), and it automatically prunes unpromising trials for faster results.

Specifically, it leverages a history of past trials to inform subsequent hyperparameter selections. By analyzing this historical data, it identifies promising regions of the hyperparameter space to explore further. This iterative refinement is driven by a Bayesian optimization algorithm, which enables efficient and effective hyperparameter optimization.

For this project, we defined a maximum of 50 trials for the hyperparameter optimization process.

3.2. Benchmark

In order to obtain a benchmark model and initially check the performance of our chosen features in the text classification process, we train and evaluate a simple Logistic Regression model.

On our first trial, we utilized as input to the model only the text embeddings and the results can be seen below. As expected, the performance is not optimal as it achieves an accuracy of 58% and a macro f1-score of 59%.

```
==========Confusion Matrix==========

[[ 6  0  1  1  0  0  0  0  0]
 [ 0  9  1  2  0  1  0  0  1]
 [ 1  1 13  6  0  2  1  1  2]
 [ 1  1  1 40  0  6  0  2  4]
 [ 0  0  5  1  5  0  0  0  0]
 [ 0  0  2  8  2 13  0  1  2]
 [ 0  0  2  2  0  1  5  0  0]
 [ 0  0  1  3  0  1  0  5  0]
 [ 0  1  1  6  0  1  0  0 10]]

==========Classification Report==========

              precision    recall  f1-score   support

           0       0.75      0.75      0.75         8
           1       0.75      0.64      0.69        14
           2       0.48      0.48      0.48        27
           3       0.58      0.73      0.65        55
           4       0.71      0.45      0.56        11
           5       0.52      0.46      0.49        28
           6       0.83      0.50      0.62        10
           7       0.56      0.50      0.53        10
           8       0.53      0.53      0.53        19

    accuracy                           0.58       182
   macro avg       0.63      0.56      0.59       182
weighted avg       0.59      0.58      0.58       182
```

We should also note that we examined the Logistic Regression performance using SVD on the text representations, to reduce the dimensionality of the text features before feeding

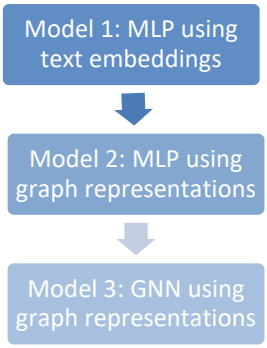
[3] https://optuna.org/

them into the logistic regression classifier and reduce any possible noise that is present in our data. The number of features we tested varied from 100-200, which they explained over 95% of the variance. However, the performance was not improved, so the SVD was not further used.

3.2. Final model

In this section, we present the architecture of our modeling approach, which produced the model with the best performance.

Specifically, this architecture sequentially utilizes three different models. The output of each model serves as the input to the subsequent model, with additional features incorporated at each stage. A high – level representation can be seen below:



This iterative enhancement aims to progressively improve the performance by leveraging the combined strength of text embeddings and the node connectivity metrics. Further details on each model and the specific features introduced at each stage are explained below.

Model 1: ***MLP using text embeddings***

Starting the modeling process, we decide to train a ***MLP model*** that takes as input features the **text embeddings** produced using the Sentence Tokenizer, of size (768,). We firstly split the data into 90% train and 10% validation sets, in order to evaluate the generalization power of each model. Then, we define the search space to be used in the *Optuna* framework, where we define the max trials as 50:

Parameter	*Range*
Number of layers	1 - 4
Hidden units	256 - 1024, step =128
Dropout rate per layer	0.1 – 0.5, step = 0.1
Batch size	32, 64, 128
Learning rate	1e-2, 1e-3, 1e-4

For the training process, we also utilize *Early Stopping* in order to avoid the overfitting of the model, using as patience 5 epochs and then we train the model for 100 epochs. The model with the best hyper-parameters is the following, which achieves a train loss of 0.8964 and a validation loss of 1.0685:

Parameter	*Optimal Values*
Number of layers	2
Hidden units – Layer 1	896
Dropout rate – Layer 1	0.2
Hidden units – Layer 2	384
Dropout rate – Layer 2	0.4
Batch size	64
Learning rate	1e-4

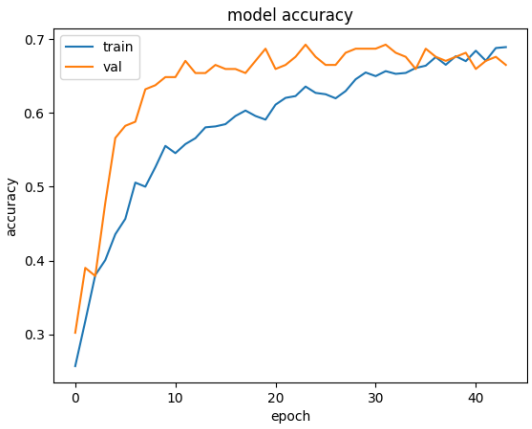
Model: "sequential_50"

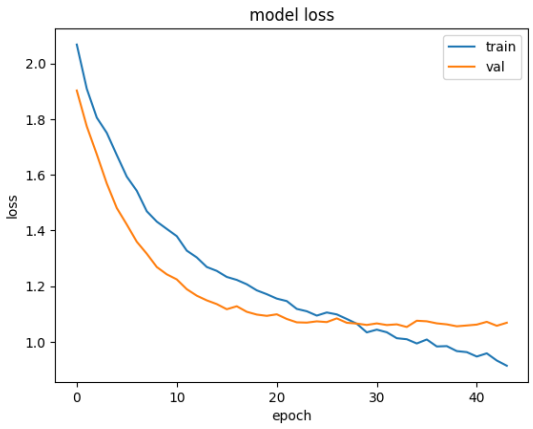
Layer (type)	Output Shape	Param #
dense_107 (Dense)	(None, 896)	689,024
dropout_107 (Dropout)	(None, 896)	0
dense_108 (Dense)	(None, 384)	344,448
dropout_108 (Dropout)	(None, 384)	0
Output (Dense)	(None, 9)	3,465

Total params: 1,036,937 (3.96 MB)

Trainable params: 1,036,937 (3.96 MB)

Non-trainable params: 0 (0.00 B)





From the above plots we can see that after around 20 epochs, the training and validation loss curves start to converge, indicating that the model is stabilizing. However, the training

loss continues to decrease, while the validation loss shows a slight increase towards the end, indicating potential overfitting. Besides the Early Stopping mechanism we used, which worked and stopped the training at around 40 epochs, we could utilize Regularization methods or increase the dropout rate.

Using the validation dataset, we evaluate the trained MLP model, which leads to the following results. As we can see, this model already outperforms the Logistic Regression used as baseline, which was expected.

```
==========Confusion Matrix==========

[[ 5  0  0  1  0  2  0  0  0]
 [ 0  9  1  2  0  1  0  0  1]
 [ 0  0 19  3  0  1  1  2  1]
 [ 1  2  3 46  0  1  0  0  2]
 [ 0  0  4  0  6  1  0  0  0]
 [ 0  0  2  9  1 14  0  0  2]
 [ 0  0  2  2  0  1  5  0  0]
 [ 0  0  0  2  0  2  0  6  0]
 [ 0  0  1  5  0  0  0  0 13]]

==========Classification Report==========

              precision    recall  f1-score   support

           0       0.83      0.62      0.71         8
           1       0.82      0.64      0.72        14
           2       0.59      0.70      0.64        27
           3       0.66      0.84      0.74        55
           4       0.86      0.55      0.67        11
           5       0.61      0.50      0.55        28
           6       0.83      0.50      0.62        10
           7       0.75      0.60      0.67        10
           8       0.68      0.68      0.68        19

    accuracy                           0.68       182
   macro avg       0.74      0.63      0.67       182
weighted avg       0.69      0.68      0.67       182
```

Model 2: MLP using the output of Model 1 combined with graph representations

After training the best-performing MLP on the first step, we obtain the probability distribution it produces for each input, which has a size of (9,) corresponding to our 9 classes. This probability distribution reflects the confidence of this model in assigning each input to the classes. In order to further enhance our model's performance, we introduce the additional graph-based features.

Specifically, we create the graph representation as described in the respective section above. This representation consists of feature vectors of size 128, derived from Word2Vec application on random walks, along with the three-node metrics: in-degree, out-degree, and average neighbors' degree and the (1,) vector that indicates the partition. These combined vectors, totaling a size of 132 (128 + 3+1), capture the structural properties and connectivity information of the nodes within the graph.

This **combined input** for the second MLP, is thus formed by **concatenating the probability distribution from Model 1 with the graph representation vectors**. The purpose of using the probability distribution from Model 1 as an input to Model 2 is twofold. Firstly, it enables Model 2 to build upon the initial classification, leveraging the probabilistic information that

reflects the initial model's confidence in class assignments. Secondly, by integrating this probabilistic data with graph-based features, Model 2 can refine its understanding of the input data, potentially capturing more complex relationships and dependencies that the initial model might have missed.

Following, using the approach described earlier, we scale the input vectors since the features are on a different scale, where the probabilities range from 0 to 1 and the graph representation features are much larger numbers. We tried both no scaled and scaled input data, and we saw that scaling helped in the convergence of algorithms and resulted in better results (val loss and accuracy).

Again, as of Model 1, we define the same search space for the Optuna framework to utilize and find the best hyper – parameters, by performing 50 trials at maximum and we utilize *Early Stopping.* The model with the best hyper-parameters is the following, which achieves a train loss of 0.2842 and a validation loss 0f 0.7445:

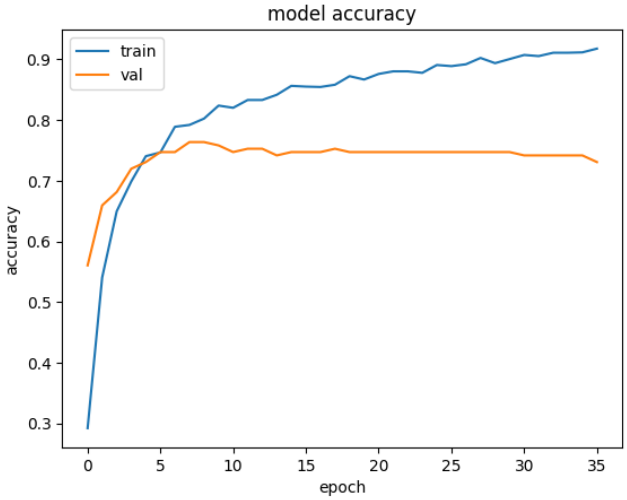
Parameter	***Optimal Values***
Number of layers	1
Hidden units	896
Dropout rate	0.3
Batch size	128
Learning rate	1e-4

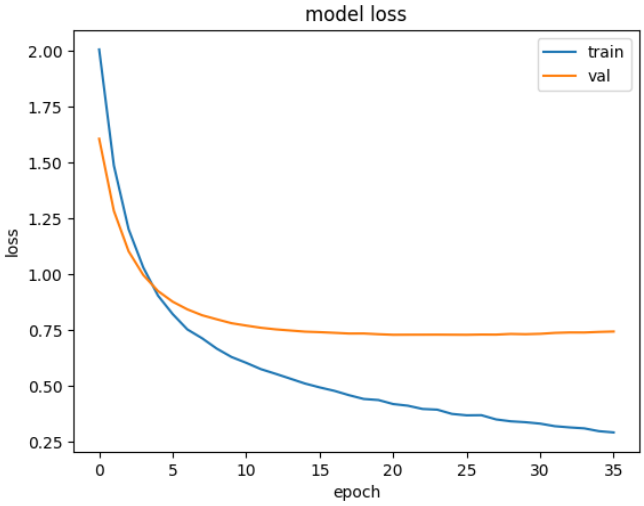
Layer (type)	Output Shape	Param #
dense_190 (Dense)	(None, 896)	127,232
dropout_190 (Dropout)	(None, 896)	0
Output (Dense)	(None, 9)	8,073

Total params: 135,305 (528.54 KB)

Trainable params: 135,305 (528.54 KB)

Non-trainable params: 0 (0.00 B)





From the plots we can see that the validation loss starts to level off around the 5th epoch and shows minimal improvement after that, indicating that the model may start overfitting to the training data around this point. The training loss continues to decrease steadily throughout the training period, suggesting that the model is continuing to learn the specifics of the training data. The gap between the training loss and the validation loss widens as the epochs progress. This widening gap is a strong indicator of overfitting, where the model performs well on training data but does not generalize as well to unseen validation data. To resolve this, we could add regularization methods or widen the search space of the hyper parameters, by increasing the possible dropout rates.

Using the validation dataset, we evaluate the best MLP model produced, which leads to the following results. As we can see, this model achieves better performance that the first MLP model, since it effectively integrates the probability distribution from Model 1 with the enriched graph representations. The enhanced feature set enables the model to capture more complex patterns and relationships, thereby providing a more comprehensive understanding and classification of the input data.

	precision	recall	f1-score	support
0	1.00	0.62	0.77	8
1	1.00	0.86	0.92	14
2	0.68	0.63	0.65	27
3	0.75	0.85	0.80	55
4	0.88	0.64	0.74	11
5	0.57	0.61	0.59	28
6	0.80	0.40	0.53	10
7	0.82	0.90	0.86	10
8	0.78	0.95	0.86	19
accuracy			0.75	182
macro avg	0.81	0.72	0.75	182
weighted avg	0.76	0.75	0.74	182

We could mention that:

- We also developed and tested one MLP model (instead of 2) that integrates text embeddings and graph-based features. Specifically, we truncated the text embeddings to a 100-dimensional space and combined them with 132 graph-based features, creating a comprehensive feature set to capture both textual and structural information. We ultimately decided to abandon this model, which combined truncated 100-dimensional text embeddings with 132 graph-based features, as its performance on the submission test was slightly lower compared to the alternative model.

	precision	recall	f1-score	support
0	0.70	0.88	0.78	8
1	0.92	0.79	0.85	14
2	0.59	0.59	0.59	27
3	0.72	0.84	0.77	55
4	0.78	0.64	0.70	11
5	0.45	0.46	0.46	28
6	0.71	0.50	0.59	10
7	0.82	0.90	0.86	10
8	0.85	0.58	0.69	19
accuracy			0.69	182
macro avg	0.73	0.69	0.70	182
weighted avg	0.69	0.69	0.68	182

- Furthermore, we tested a Logistic Regression model using same features set. The model's performance improved significantly with these enhanced features compared to the benchmark Logistic Regression and achieves accuracy of 68%; however it remains worse than the MLP model 2.

	precision	recall	f1-score	support
0	0.71	0.62	0.67	8
1	0.85	0.79	0.81	14
2	0.63	0.70	0.67	27
3	0.76	0.67	0.71	55
4	0.89	0.73	0.80	11
5	0.50	0.50	0.50	28
6	0.43	0.30	0.35	10
7	0.75	0.90	0.82	10
8	0.63	0.89	0.74	19
accuracy			0.68	182
macro avg	0.68	0.68	0.67	182
weighted avg	0.68	0.68	0.67	182

Model 3: GCN using the output of Model 2 combined with graph representations

As a final model, we decided to use a **Graph Convolutional Neural Network (GCN)** model, which is suitable for data similar to those used in this project, since they are a class of neural networks specifically designed to work with graph-structured data.

More analytically, **GCNs** is a powerful technique for extracting and summarizing node information within a graph structure. It is derived from the convolution operation used in Convolutional Neural Networks (CNNs), which are commonly applied to image data.

In GCNs we create features for each node, and we store them in a feature matrix for efficiency purposes. Like almost every Graph Deep Learning Architecture, the GCN also has two key processes: **Message passing** and **Update.** In GCNs, the message passing, and aggregation step involves aggregating features from neighboring nodes and the node itself (a weighted sum of the feature vectors of the neighbors). To ensure appropriate scaling, we

use a normalized adjacency matrix. The update rule combines the aggregated features with a learnable weight matrix and applies a non-linearity function.

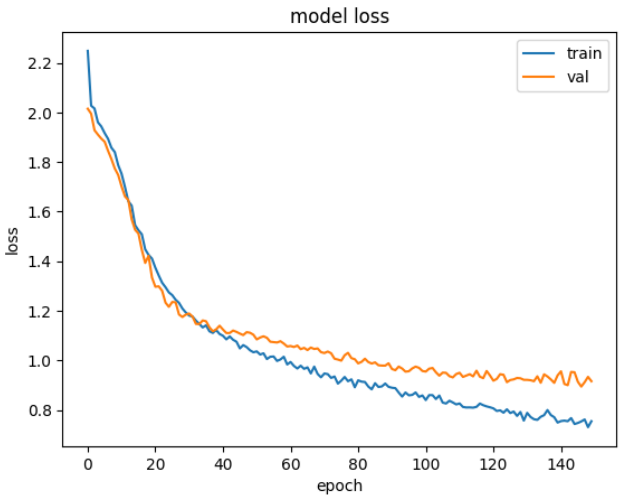
The combination of these steps in multiple layers allows GCNs to effectively capture the structural information of the graph and learn meaningful node embeddings.

More specifically, regarding the **input features**, we should note that since the entire graph is used as data, features should be created for all nodes. Therefore, for the nodes where we have data available, we utilize the **probability vector** of size (9,) which is produced **from the Model 2** and **the 128-dimensional vectors produced from the random walks using Word2Vec** (as explained in the respective section). For the nodes where we do not have data, we generate **9 random values** between 0 and 1, and also use the **128-dimensional vectors the random walks using the Word2Vec model**. The use of those embeddings which contain Topological Context is a better alternative to the creation of random features. With our approach, the model has information even before the beginning of its training.

The **architecture** of the GCN consists of **two GCN layers** followed by a **fully connected layer**. In the forward pass, the input features propagate through the first GCN layer, followed by a **ReLU** activation function and dropout for regularization. Then the output passes through the second GCN layer with similar activation and dropout. Finally, the output is fed into a linear layer to produce class scores, which are then passed through a log-softmax function to obtain output probabilities over classes. We train the model for 150 epochs. The hyper-parameters used to train the model are the following:

Parameter	*Values*
Number of layers	2
Hidden units – Layer 1	64
Hidden units – Layer 2	128
Dropout value	0.4
Learning rate	0.02

The results can be seen below. The model achieves a train loss of 0.7550 and a validation loss of 0.9162. From the plot of train and validation loss, we can see that the issue of overfitting seems to be resolved or at least is less present.



Using the validation dataset, we evaluate the model produced, which leads to the following results. As we can see, this model achieves accuracy at 71% and a f1-score of 68%.

	precision	recall	f1-score	support
0	0.57	0.50	0.53	8
1	0.83	0.71	0.77	14
2	0.73	0.70	0.72	27
3	0.77	0.85	0.81	55
4	0.82	0.82	0.82	11
5	0.54	0.50	0.52	28
6	0.44	0.40	0.42	10
7	0.75	0.90	0.82	10
8	0.78	0.74	0.76	19
accuracy			0.71	182
macro avg	0.69	0.68	0.68	182
weighted avg	0.71	0.71	0.71	182

We should notice that even though the accuracy is slightly less than the previous MLP model, the overfitting issue seems to be better in this model.

By utilizing the probability distributions produced from this model for the unknown test set, we achieve a **Private loss of 0.9196** and a **Public loss of 0.8620** on the Kaggle submission.

3.3. Model trials

- **Modeling trials**:
  - We initially tested models using different feature sets independently:
    - MLP model using the text embeddings alone.
    - MLP model using graph-based features alone, both with and without Louvain community labels.

- **Combined Feature Models**:
  - We experimented with combining text embeddings and graph-based features:
    - MLP model using the full text embeddings (768,) combined with graph-based features.
    - MLP model using the truncated text embeddings (100,) combined with graph-based features.

- **Feature Scaling:**
  - For every feature set combination, we evaluated the model performance both with and without feature scaling to assess its impact on results.

Best Model Identification based on the final leaderboard results

After the private results were revealed on Kaggle post-competition, we observed that our best model achieved a score of 0.8735. However, this model was not selected for evaluation because its public score on a sample of the test set (0.9881) was lower than our best submission's public score (0.8929).

The best predictions were the output of the second MLP model, using the described approach, combining the predicted probabilities with the graph-based features of size (141,)

4. Further Improvements

For further improvements, several strategies could be explored to address observed limitations and enhance model performance.

- Firstly, given the indications of overfitting observed in our models, implementing regularization techniques such as batch normalization layers in the MLP architectures could help mitigate this issue.
- Additionally, to address discrepancies between validation and test losses, techniques such as cross-validation could be employed for fine-tuning, providing a more robust assessment of model performance across different data splits.
- Moreover, to handle the imbalanced dataset, approaches like data augmentation or random sampling could be employed to ensure better representation of minority classes.
- Exploring advanced feature engineering methods such as topic modeling to extract topic distributions from texts, instead of relying solely on text model's predicted probability distributions, could enhance the model's ability to capture nuanced information.
- Lastly, for improving graph-related features, incorporating additional measures such as PageRank or other centrality metrics could provide richer representations of node importance and connectivity within the graph structure.