GRAPH EMBEDDINGS FOR NATURAL LANGUAGE PROCESSING

INF580 - Mathematical Programming

March 16, 2024

Guilhereme Vieira Manhaes
Artur Cesar Araujo Alves

CONTENTS

1 Introduction **3**

2 Methods **3**
2.1 Graph of Words . . . 3
2.2 Token Similarity . . . 3
2.3 Isomap . . . 4
2.4 Non Linear Approach . . . 4
2.5 Linear Approach . . . 5
2.5.1 Linearization . . . 5
2.5.2 Relaxation . . . 5
2.5.3 Projection . . . 5

3 Results **6**
3.1 Summary . . . 6
3.1.1 Similarity . . . 6
3.1.2 Training Size . . . 6
3.1.3 NLP . . . 6
3.1.4 Analysis . . . 7

1 INTRODUCTION

Graph Embedding is a known subject in mathematical programming. The problem can be stated as follows: Given a weighted graph $G = (V, E \subset V \times V, d : E \to \mathbb{R}^+)$ we want to produce an embedding of dimension $k \in \mathbb{N}$, $\phi : V \to \mathbb{R}^k$ that better approximate d in the following sense:

$$\forall (u, v) \in E, d(u, v) \approx ||\phi(u) - \phi(v)||$$

The generation of embedding for elements in a Euclidean space is useful in many domains of knowledge, mainly Natural Language Processin. One way to achieve this via these methods is to encode a corpus of text as a Graph and encode distance between its nodes using known similarities from language processing like the Path Similarity . This method of encoding text has been show to produce satisfactory results in text mining and language understanding, as pointed out by Vazirgiannis.

In this project we will combine these two fields of study to develop and evaluate methods to generate word embeddings using graph embeddings.

2 METHODS

2.1 Graph of Words

There are several ways to immerse text into graph structures. One of these approaches is the graph-of-words (GoW), that captures word relations based on their proximity in a sentence. Let $S = w_1 w_2 \dots w_n$ be a (already tokenized) sentence of length n. To construct a GoW of radius r of this sentence ($GoW_r(S)$) we first set $V = w_1, \dots, w_n$ the vertices. To construct the set of edges E we choose the words that are positioned closely in the sentence. In other words:

$$(w_i, w_j) \in E \iff |i - j| \leq r$$

2.2 Token Similarity

The graph generated by the GoW algorithm provides structure but it lack a definition of distance between words that convey any semantic meaning. To do that, one can enrich the graph by providing it with a function $s : V \times V \to \mathbb{R}^+$ that will provide that measure. This function is often called similarity and there are several means of constructing a function with this signature.

One very common method implemented by NLTK on the WordNet dataset is the Path Similarity. This functions takes into consideration the syntactic distance between tokens in a way that places tokens with meanings next to each other. There are several other distances implemented for this data set such as the Wu-Palmer similarity and the Leacock-Chodorow similarity. More information on this can be found in the official documentation

2.3 Isomap

The Isomap is an approach that relies not on mathematical solvers but on a very strong theorem of Universal Isometric Embedding. According to this Theorem, given a finite metric space $(X = \{x_1, \ldots, x_n\}, d)$ one can generate an embedding function $\phi : X \times X \to \mathbb{R}^+$ by defining trivially

$$\phi(x) = (d(x, x_1), d(x, x_2), \ldots, d(x, x_n))$$

This function generates embeddings that accurately depict the metric d for the l_∞ norm. In other words:

$$||\phi(x) - \phi(y)||_\infty = d(x, y)$$

However, this is only useful on metric spaces and our graph structure does not posses a metric like this out of the box. Given $G = (V, E, \delta)$ a connected graph, one common approach to fabricate this function is to define $d(u, v) = \text{shortestPath}_G(u, v)$. This can be computed using the Floyd–Warshall algorithm with the initial distance function δ. This a metrix space from which we can extract an embedding.

One other impracticality of this model is that the embedding space is very sparse, of dimension n. To extract a dense representation of dimension $k < n$ we can utilise a dimensionality reduction method like PCA. the PCA decomposition allows us to decompose $X_{n,n} = P\Lambda Q$. By extracting the k largest eigenvalues of Λ (Λ^k) we can construct $T_{n,n} = \Lambda^k Q^k$ that can be used as an approximate embedding of our graph.

2.4 Non Linear Approach

A more direct approach to solving the DGP problem involves solving a system of quadratic equations, but this is often computationally unfeasible. An alternative is to reformulate the problem by minimizing the mean squared error:

$$\min_{\phi} \sum_{u,v} \left(\|\phi(u) - \phi(v)\|^2 - w(u, v)^2 \right)^2$$

However, this still requires solving a nonlinear programming problem (quadratic problem), which is highly computationally demanding. This approach is more practical when combined with a well-initialized embedding ϕ.

2.5 Linear Approach

2.5.1 • Linearization

To linearize the DGP, we can reformulate it using the scalar product $P : E \to \mathbb{R}^+$, where E represents the embedding space. This reformulation involves equating the squared Euclidean distance between embeddings to the given pairwise distances. Specifically:

$$\|\phi(u) - \phi(v)\|^2 = P(u,u) + P(v,v) - 2P(u,v)$$

To ensure the viability of the reformulation, we must confirm that a potential embedding can achieve the predicted scalar product values P. Formally, there must be a ϕ such that $P = \phi\phi^T$.

2.5.2 • Relaxation

One way to relax this requirement, is to enforce the matrix P to be positive semi-definite, that is, instead of forcing the difference $P - \phi\phi^T$ to have zero eigenvalues we only require them to be non negative.

To proceed with this approach, we have two options:

1. Utilize a SDP (Semi-Definite Programming) solver with the condition $P \succeq 0$.
2. Enforce P to be diagonal dominant, as diagonal dominant matrices are PSD.

A suggested approach, introduced in class, involves reformulating the problem using a push-and-pull method applied to both the objective function and the constraint associated with the distance matrix. This leads to a Diagonal Dominant Problem (DDP) formulation as follows:

$$\begin{cases} \min\limits_{(u,v)\in E} P(u,u) + P(v,v) - 2P(u,v) \\ P(u,u) + P(v,v) - 2P(u,v) \geq w(u,v)\,, \forall (u,v) \in E \\ \sum\limits_{u \neq v} T(u,v) \leq P(u,u)\,, \forall u \in V \\ -T \leq X \leq T \end{cases}$$

Here, the final two constraints enforce the diagonal dominance property on the solution $\bar{P}$.

2.5.3 • Projection

Finally, to retrieve the embedding ϕ from $\bar{P}$ with the correct rank, we can either factor the solution with PCA, reducing the embedding dimension to the desired one K, or use Barvinok's naive algorithm to obtain a decent approximation:

1. Sample a matrix y from $N^{n\times K}(0, \frac{1}{\sqrt{K}})$

2. Factor $\bar{P}$, i.e., obtain $\Lambda_{n\times n}$ such that $\Lambda\Lambda^T = \bar{P}$
3. Reduce the factor dimension to K: Λy

3 RESULTS

Probably due to the poor amount of data we can feed the different approaches described and still get a reasonable execution time, we unfortunately could not obtain visually meaningful embeddings in our experiments as in figure 1.

Yet, we conducted a series of experiments to investigate the impact of various factors on the performance of selected algorithms. Specifically, we examined the influence of the similarity function, training sizes, and the incorporation of the NLP over an initial solution. Our experimentation involvesd employing the Isomap approach and the DDP approach, utilizing both PCA and Barvinok's naive algorithm as projection methods to retrieve embeddings in 3 dimensions, which we compare using the mean distance error:

$$\frac{1}{|E|}\sum_{(u,v)\in|E|}|\|\phi(u)-\phi(v)\|-d(u,v)|$$

In these experiments, we utilized the AMPL Python API, employing `cplex` as the linear solver and `ipopt` as the nonlinear solver. Additionally, for text processing, the `brown` corpus from NLTK was utilized. Our code implementation can be found **here**.

3.1 Summary

3.1.1 • Similarity

Fixing training size at 50 sentences and using NLP to improve the solution, we test the 3 algorithms over two similarity scores from `nltk`: *Path* and *WuPalmer*. Results in table 1.

3.1.2 • Training Size

Fixing similarity to be *Path* and using NLP to improve the solution, we test the 3 algorithms over 3 training sizes: 10, 50 and 100 sentences from the `brown` corpus. Results in table 2.

3.1.3 • NLP

Fixing similarity to be *Path* and training size at 50 sentences we test the 3 algorithms over the presence or absence of the NLP to improve the solution. Results in table 3.

3.1.4 • Analysis

The findings indicate:

1. All three models performed better with *Path* as the similarity score for graph embeddings.
2. Larger training sizes improved model fitting.
3. NLP notably enhanced solutions only with the Isomap approach.

Comparing the three approaches, using Barvinok's naive algorithm and PCA projection in DDP seems to be nearly equivalent, at least in three dimensions. Furthermore, Isomap showed to be less effective than these methods.

APPENDIX

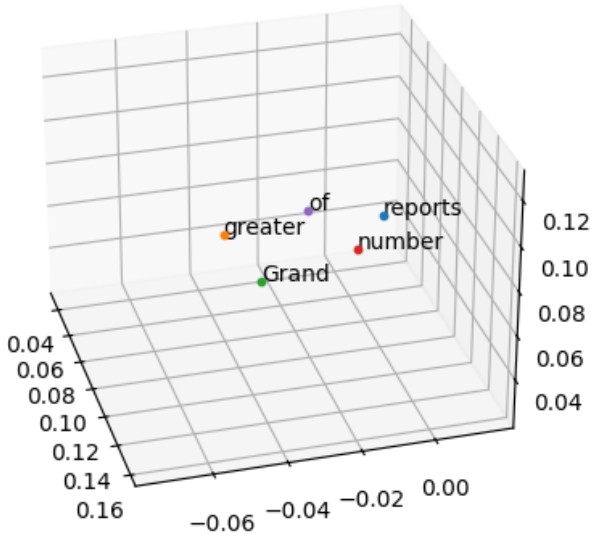

Figure 1: Example of embedding found in 3d for a random sample from the corpus

Algorithm	Similarity	
	Path	Wup
ddp_{pca}	1.743e-3	3.654e-3
$\text{ddp}_{\text{barvinok}}$	1.743e-3	3.654e-3
isomap	2.179e-2	4.494e-2

Table 1: Comparison of algorithms using different similarities

Algorithm	Training Size		
	10	50	100
ddp_{pca}	3.922e-2	1.743e-3	1.284e-3
$ddp_{barvinok}$	3.921e-2	1.743e-3	1.284e-3
isomap	3.922e-2	2.179e-2	1.880e-2

Table 2: Comparison of algorithms using different training sizes

Algorithm	NLP	
	Present	Absent
ddp_{pca}	1.743e-3	1.743e-3
$ddp_{barvinok}$	1.743e-3	1.743e-3
isomap	2.179e-2	4.344e+0

Table 3: Comparison of algorithms with and without NLP to improve the solution