BUILDING DETERMINISTIC NODE REPRESENTATIONS FROM PERSONALIZED PAGERANK SEQUENCES

Gabriel Augusto Amim Sab

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Janeiro de 2021

BUILDING DETERMINISTIC NODE REPRESENTATIONS FROM PERSONALIZED PAGERANK SEQUENCES

Gabriel Augusto Amim Sab

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Daniel Ratton Figueiredo

Aprovada por: Prof. Daniel Ratton Figueiredo
Prof. Valmir Carneiro Barbosa
Prof. Pedro Olmo Stancioli Vaz De Melo

RIO DE JANEIRO, RJ – BRASIL
JANEIRO DE 2021

Sab, Gabriel Augusto Amim

Building Deterministic Node Representations from Personalized PageRank Sequences/Gabriel Augusto Amim Sab. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIV, 86 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 56 – 58.

1. Complex Networks. 2. Network Science. 3. Graphs. 4. Graph Embeddings. 5. Embeddings. 6. Personalized PageRank. 7. PageRank. 8. PPR. I. Figueiredo, Daniel Ratton. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Esta dissertação é dedicada à pesquisa científica brasileira, em um momento de grande descrédito e desvalorização. Cada contribuição feita ajuda a mantê-la viva, e espero ter contribuído de alguma forma para isso.

Agradecimentos

Agradeço a todas as pessoas próximas que me incentivaram nos momentos em que estive desanimado. A todos os colegas que se dispuseram a ouvir sobre meu trabalho e contribuir com ideias e perguntas. A todos os amigos que se dispuseram a ler meu texto e fornecer sugestões e críticas. Por fim, agradeço ao meu orientador, Daniel, pela compreensão, incentivo e atenção fornecidos ao longo desse tempo.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CONSTRUINDO REPRESENTAÇÕES DETERMINÍSTICAS PARA VÉRTICES A PARTIR DE SEQUÊNCIAS GERADAS COM PERSONALIZED PAGERANK

Gabriel Augusto Amim Sab

Janeiro/2021

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Um dos avanços recentes mais importantes na área de aprendizado de máquina foi a capacidade de traduzir dados de alta dimensionalidade para espaços vetoriais de dimensionalidade relativamente baixa, uma técnica conhecida como *embedding*. Devido à complexidade e diversidade de redes reais, *embeddings* surgem como um caminho para representar redes por meio de vetores, de preferência capturando informação relevante, e portanto permitindo o uso de modelos clássicos de aprendizado de máquina com dados de redes reais. Uma propriedade interessante é a identidade estrutural dos vértices de uma rede, ou a função que um vértice exerce na rede, independente de sua identidade (rótulo) e da identidade de seus vizinhos na rede. De fato, abordagens recentes geram *embeddings* de vértices que refletem as suas identidades estruturais. Este trabalho propõe um método para construir representações determinísticas de vértices a partir de sequências geradas com o algoritmo de *Personalized PageRank* (PPR), que são únicas para cada identidade estrutural, funcionando como uma "assinatura" da função estrutural dos vértices. A capacidade do método de diferenciar as identidades estruturais presentes em diferentes redes é avaliada empiricamente. Também é avaliado o seu desempenho em uma tarefa de classificação de vértices dependente das identidades estruturais dos vértices, comparando-o a métodos propostos anteriormente.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

BUILDING DETERMINISTIC NODE REPRESENTATIONS FROM PERSONALIZED PAGERANK SEQUENCES

Gabriel Augusto Amim Sab

January/2021

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

One of the most important recent advances in machine learning is the ability to translate high-dimensional data into relatively low-dimensional vector spaces, a technique known as embedding. Due to the complexity and diversity of real networks, embeddings arise as a way of representing networks as vectors, hopefully capturing relevant information, and therefore creating the possibility of using classic machine learning models on real graph data. An interesting property is the structural identity of network nodes, or the role a node plays in the network, regardless of its identity (label) and the identity of its neighbors. Indeed, recent approaches generate node embeddings that reflect their structural identity. This work proposes a method for building deterministic node representations from Personalized PageRank (PPR) sequences that are unique to each structural identity, effectively acting as a signature of that structural role. We empirically evaluate the capacity of the method to differentiate the structural roles in different networks, as well as its performance on a node classification task that is dependent on the structural identities of nodes, while also comparing with prior approaches.

Contents

List of Figures

List of Tables

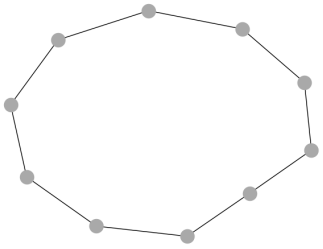
Chapter 1

Introduction

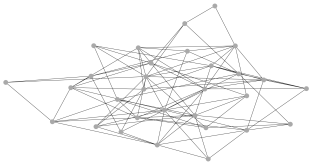
Many aspects in society and nature can be modeled by a set of pairwise relations, essentially forming a network. Social networks model friendships, professional connections, followership and many other types of human interactions. Transport networks model the way vehicles, people, cargo, information, water, energy and a myriad of other things move around a city, country or even the whole world. Biological networks model the interactions between proteins, cells, organisms, species and biomes.

When the structural features of a network are not trivial, meaning they are not easily captured by a simple network model, it is named a complex network. In other words, their edges (i.e., the pairwise relations) do not follow either a purely random or a purely regular pattern. In that sense, most real networks are complex by nature. Figure 1.1 shows examples of a regular network, a random network and a real network. Network Science is an area of scientific research with the purpose of studying complex networks so that their structures and mechanisms are better understood, in order to model and make predictions for nodes, links and entire networks with an acceptable level of confidence. The area has gained a lot of traction recently with the emergence of datasets of very large real networks from various domains, including online social networks.

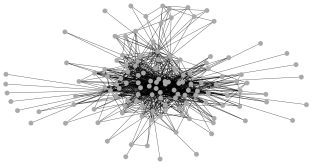
Recently, machine learning, a field of artificial intelligence dedicated to the study



(a) Cycle network



(b) G(30, 0.2) Erdős–Rényi random network



(c) Real network (Brazilian airports)

Figure 1.1: Examples of network structures

of computer algorithms that learn from data, has also come under the spotlight, with many new theoretical and applied works appearing daily. One of its recent advances is the ability to translate high-dimensional data into relatively low-dimensional vector spaces, a technique known as embeddings. This technique has led to previously unseen levels of performance, and was first applied to natural language processing, surpassing what was considered the state-of-the-art. Since then, embeddings have been used in several other areas, including networks.

As the structure of complex networks is not faithfully captured by any currently known network models, embeddings can be used to condense or summarize these structures into vectors, hopefully capturing relevant information. Since vector spaces are the most prevalent data model for machine learning models, this opens the possibility of applying traditional models on real graph data. Several studies have been successfully conducted on this topic in recent years, and several more are currently under development. Clearly, this approach has shown to be quite powerful for many different tasks, such as node classification and link prediction.

Most existing works on the topic, such as node2vec [1] and GraphSAGE [2], are more focused on capturing homophilic relations, where edge points tend to have similar traits and preferences. However, another interesting property is the structural identity of nodes, or the role nodes play in the network. In other words, the local structure around each node determines the node's identity, and not its label. Thus, two nodes that have different neighbors (in terms of labels) could have very similar roles in the network. struc2vec [3] is a pioneering framework in this direction.

This work proposes a method for building deterministic node representations from Personalized PageRank (PPR). The PageRank [4] algorithm is a well-known algorithm proposed by the founders of Google to better rank relevant web pages, in order to improve search results. Its personalized version alters the resulting ranks in order to favor pages that are closer to a group of target pages, or even a single page. However, one aspect of PPR that has not been explored is the values that the target node assumes during the iterative procedure to compute the PageRank. As the algorithm is purely based on the structure of the network, this sequence captures structural identity, effectively acting as a signature for the structural role. We leverage this characteristic to generate sequences at several levels of personalization for each node.

Unlike the other node embedding methods discussed in this work, which have an inherent random factor, the PPR method here proposed is deterministic, meaning it is consistent across different executions on the same dataset. The method is also capable of generating a representation for a single network node, if desired, without the need of generating a representation for every node (or randomly sampled subsets of nodes) in the network, which is the case for most other methods. Finally, the

representations for nodes that belong to the same automorphism group are exactly the same, which is something even struc2vec, whose focus is on structural identity, does not perfectly achieve.

Thus, this work makes the following contributions:

- A novel embedding method based on PPR to capture the structural identity of nodes. The main insight is to leverage the sequence of PPR values that emerges from its iterative computation. The method is deterministic and does not require jointly embedding multiple nodes. Details are presented in Chapter 4.

- A methodology for evaluating consistency and robustness of node embedding methods, using metrics as described in Chapter 3.

- An empirical evaluation of the consistency and robustness of the representations generated by two prominent embedding methods (GraphSAGE [2] and struc2vec [3]) using three different real networks. Details are presented in Chapter 3.

- An empirical evaluation of the proposed method on the same three different real networks, indicating the capacity of the method in differentiating the many structural roles of network nodes. Also, an evaluation of the parameters in the resulting representations, as well as its robustness to edge removal. Details are presented in Chapter 4.

- An evaluation and comparison of the PPR embeddings when used for node classification, considering three real networks (airport networks) and node labels that have a strong relationship to their structural role. The performance in terms of classification accuracy shows the potential of the PPR embeddings on learning models where the input features have a temporal component, such as Recurrent Neural Networks. Details are presented in Chapter 5.

Finally, while PPR embeddings did not outperform struc2vec on the classification task considered, the methodology showed its potential, in particular since the methodology is fairly orthogonal to most prior approaches towards node embeddings.

Chapter 2

Related Work

This Chapter presents some of the related work concerning embeddings, a technique to represent objects in low dimensional spaces while preserving some of these objects' characteristics. We will first briefly discuss the word2vec method, since it inspired most of the subsequent work on embeddings, including graph and node embeddings. Then, we will discuss the inner workings and the applicability of some of the aforementioned methods for generating node embeddings for graphs. We then briefly discuss how embeddings are used as features in node classification tasks. We describe graph traversal methods (e.g., BFS and random walks), as well as the Personalized PageRank (PPR) algorithm and the role of its fundamental parameter (i.e., return probability). In particular, PPR is the core of the methodology proposed in this work.

2.1 word2vec

word2vec [5, 6] is a method proposed by Mikolov et al. to learn vector representations of words in a way that captures their semantic similarities. These representations lie in a space whose dimensionality is lower than that of the original representation. The method uses one of two neural network model architectures: Continuous Bag-of-Words (CBOW) or Continuous Skip-gram. The neural network is used to learn the embeddings (i.e., feature vectors) for each word.

As stated in the original word2vec paper [5], the Continuous Bag-of-Words model is similar to the feedforward Neural Net Language Model (NNLM) proposed by Bengio et al. [7]. The feedforward NNLM model maps words in a sequence to their respective feature vectors using a shared projection matrix, concatenates the sequence of feature vectors and feeds the resulting concatenated vector to a hidden layer. After that, it feeds the output of this layer to a softmax layer that calculates the conditional probability distribution over words in the vocabulary for the next word. The model maximizes the training corpus log-likelihood. In other words, the

NNLM model learns the feature vectors that lead to conditional probabilities that best match the sequences that exist in the training data. An example would be to maximize the output conditional probability of the word "graphs" for the corpus sentence "I love graphs", using the words "I" and "love" as input.

The CBOW model, however, averages the feature vectors instead of concatenating, and feeds them directly to the softmax layer, removing the hidden layer altogether. As a consequence of using an average instead of concatenation, the order in which words appear does not matter anymore. The CBOW model also uses words that appear after the target word instead of only words that come before it. Using the same example as before, the model aims to maximize the conditional probability of the output word "love", using the words "I" and "graphs" as input (in any order).

The Continuous Skip-gram model is the dual of the CBOW, as its goal is to predict the surrounding words given the feature vector representation of a word. This means that instead of averaging many feature vectors and maximizing the conditional probability of a target word, it calculates the feature vector for the target word and feeds it to several output softmax layers, maximizing the likelihood of the words that surround the target word. Using the same example as before, this model aims to maximize the conditional probabilities of the words "I" and "graphs" using the word "love" as input.

Once the network has been trained, embeddings (i.e., feature vectors) are obtained directly from the first layer of the network, which projects input words (or some deterministic encoding of words) to vector representations.

The main result of word2vec is that the vector representations that it generates can be subjected to simple algebraic operations that capture relationships between words with similar meaning. An example would be to find the word "better" for "good" when using the words "worse" and "bad" as the base pair. Basically, all it takes is to subtract the vector for "worse" from the vector for "bad", sum the result with the vector for "good", and find the word in the vocabulary whose vector representation is closest to the resulting vector (i.e., its nearest neighbor). See Figure 2.1 for a synthetic visualization of this example on a feature space with two dimensions.

2.2 Graph Traversal Methods

2.2.1 BFS and DFS

Bread-first search (BFS) and depth-first search (DFS) are deterministic graph search algorithms that traverse a graph starting from a source node, visiting new nodes as

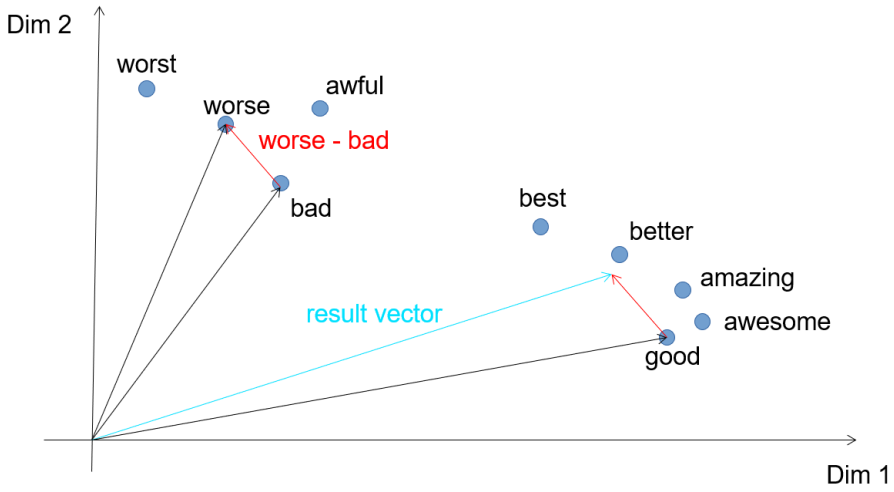


Figure 2.1: Synthetic example of algebraic operation with word2vec representations

the algorithm explores the nodes that have been previously discovered. While the breadth-first search explores all the immediate neighbors before exploring any other discovered node, the depth-first search always goes as deep as possible into the network when exploring nodes. We will use the network in figure 2.2 to illustrate. For simplicity, searches will always start from node A, and nodes will be discovered in alphabetical order.

A BFS will first explore A, discovering its immediate neighbors, nodes B, C and D. Then B is explored and its immediate neighbors are discovered. So nodes A, E and F are discovered, and A would be explored, if it had not already been. C is explored, instead, visiting A and F. Since A has already been explored, D is then explored, and so on. Node G will only be discovered once F is explored. This leads to the sequence ABCDEFG of explored nodes. Another way to explain the order in which nodes are explored is to say that a BFS places discovered nodes in a first-in first-out queue for exploration.

A DFS will also start by exploring A and discovering nodes B, C and D, exploring B next. However, when B is explored and nodes A, E and F are discovered, E becomes the next node to be explored, not C (it would be A if it had not already been explored). Then, when E is explored and no new node is discovered, F is explored, and nodes B, C and G are discovered. G is explored, and so on. Node D, as A's last immediate neighbor, will be the last node to be explored. This leads to the sequence ABEFCGD of explored nodes. Another way to explain the order in which nodes are explored is to say that a DFS places discovered nodes in a last-in first-out queue (a stack) for exploration.

BFS and DFS can be used to sample node sequences on graphs. A straightforward approach is to run BFS or DFS until a sequence of a desired length has been generated.

2.2.2 Random walks

A random walk on a graph is a stochastic process that starts from an arbitrary node V_0 and traverses edges of the graph randomly in discrete time steps, according to a probability distribution over the edges. Random walks can be applied in various contexts such as node ranking, node clustering, and for generating node sequences of any desired length, which can be used to generate embeddings of graph nodes.

Formally, given a graph $G(V, E)$, the stochastic traversal process can be described as

$$P(V_t = x|V_{t-1} = u) = \begin{cases} \frac{w_{ux}}{Z}, & \text{if } (u, x) \in E \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

where the random variable V_t denotes the node at which the walk finds itself at time step t, w_{ux} is the transition weight from node u to node x, and Z is the normalizing constant, $Z = \sum_{x \in V} w_{ux}$. Note that $\frac{w_{ux}}{Z}$ is the transition probability of moving from node u to node x.

Typically, w_{ux} is equal to the weight of the edge (u, x) if the edge exists in the graph, and equal to zero, otherwise. For unweighted graphs, this means that the next node x is chosen uniformly among the neighbors of the current node. For weighted graphs, this means that the next node x is chosen among the neighbors of u proportionately to the weight of the edge (u, x). However, the value of w_{ux} is not restricted and can be chosen in a way that better serves the purpose of the walk.

We can once again use the network in Figure 2.2 to illustrate a random walk. We will assume that the graph is unweighted and that w_{ux} is equal to one. Starting from node A, the walk will choose uniformly between B, C and D. This means each one of them has a $\frac{1}{3}$ chance of being chosen. Let's say C was chosen. Now the walk will traverse the (A, C) edge and, choose between A and F, with a $\frac{1}{2}$ chance each. Let's say A was chosen, so the walk goes back to A, and once again chooses between B, C and D, with $\frac{1}{3}$ chance each. B is chosen, and so on. The walk will stop after taking a certain number of steps, generating an ordered sequence of nodes.

A second-order random walk defines probability transitions based not only on the current location of the walker, but also on its previous location. This means that, if the walk has come to node u from node s and is about to choose the next node x, the transition probability is denoted by $P(V_t = x|V_{t-1} = u, V_{t-2} = s)$ instead.

2.3 Node Embedding Methods

Since virtually all well-established regression and classification methods work with vector representations, it would be beneficial to represent nodes as vectors. How-

ever, due to the complex nature of graph connections, there is no straightforward way to represent nodes as vectors efficiently without severely limiting the scope of the information it contains. Even though graphs can be represented by adjacency matrices, such a representation is not unique to each node, instead representing the entire network rather sparsely. Also, the adjacency list of a node does not capture enough information about the structure that surrounds the node (i.e., it is limited to a 1-hop neighborhood). This led to the emergence of several techniques that aim to generate vector representations for nodes of graphs, or graph embeddings, which aim to reduce dimensionality while preserving relevant information concerning the nodes.

We now describe three node embedding methods: node2vec [1], GraphSAGE [2] and struc2vec [3]. As the topic of graph embeddings is currently popular among researchers and practitioners, several other methods have been developed, and others are under development while this work is written. A few examples are DeepWalk [8], SDNE [9], GraphWave [10], PinSAGE [11], FastGCN [12] and SSE [13]. There are already several surveys [14–16] on the topic.

2.3.1 node2vec

node2vec is a method proposed by A. Grover and J. Leskovec to learn vector representations of graph nodes with the goal of preserving the neighborhoods of nodes in a d-dimensional feature space. In order to achieve that goal, the method uses a flexible notion of what constitutes the neighborhood of a node, based on biased second-order random walks generated for each node in the graph. Because of this flexibility, this method is a generalization of previous works, such as DeepWalk [8] and LINE [17], which use more rigid definitions of node neighborhoods.

All of these methods, including node2vec, are heavily inspired by the Skip-gram model. Essentially, they represent graphs as sets of ordered node sequences and optimize a likelihood objective that preserves neighborhoods in the same way that the Skip-gram model does with sequences of words. In essence, they maximize the probability of observing a neighborhood for a node, conditioned on the vector representation of the node. That is, for a graph $G(V, E)$, these methods optimize the objective function $\max_f \sum_{u \in V} \log P(N_S(u)|f(u))$, where $N_S(u)$ is the neighborhood of u generated with sampling strategy S and $f(u)$ is the vector representation for the node u. The strategy used to generate node sequences is a key element that differentiates one method from another.

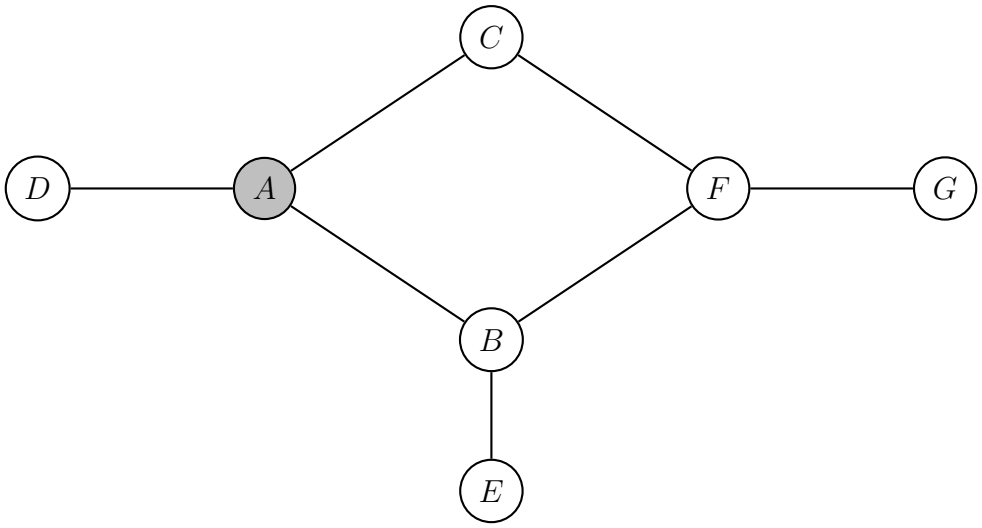


Figure 2.2: Example network, with starting node A colored gray

Sampling neighborhoods in node2vec

node2vec's sampling strategy aims to interpolate between samples generated from BFS and DFS by running biased second-order random walks. For a walk currently in node v, the bias $\alpha_{pq}(t,x)$ is a function of the shortest path distance d_{tx} between the walk's previous node t and its next node x, and also depends on two parameters p and q. It is defined by

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{cases} \tag{2.2}$$

The final edge transition probability is given by $\pi_{vx} = \alpha_{pq}(t,x) * w_{vx}$, where w_{vx} is the weight of the edge on the graph.

This means that the parameter p controls how likely the walk is to transition back to the node from which it transitioned on the previous step. For this reason, the authors call it the return parameter. Thus, this parameter can make the walk more local to the walk's starting node (when p is low), or encourage moderate exploration and avoid 2-hop redundancy in sampling (when p is high).

The parameter q controls how likely the walk is to transition to nodes that are neighbors with the current node of the walk v, but not with t (distance 2). As a consequence, when $q < 1$, the walk is more likely to transition to nodes that are farther from t than it is to transition to nodes that are close to it. This is a behavior akin to depth-first sampling. When $q > 1$, however, the walk is more likely to transition to nodes that are close to t than it is to stray farther, staying more localized, more similar to a breadth-first sampling.

The node2vec algorithm

The method works in three steps: preprocessing of transition probabilities, random walk simulations, and optimization of the neural network using stochastic gradient descent. In the preprocessing step, the edge transition probabilities π_{vx} are precomputed, so that a random walk step can be simulated in O(1) time using alias sampling. Then, r random walks of length l are simulated for each node in the network. Finally, the neighborhood sequences generated by the random walks are fed to a Skip-gram model, and contexts of size k (i.e., sub-sequences of k nodes from the resulting sequences of the random walks) are used to train feature vectors of length d.

2.3.2 GraphSAGE

GraphSAGE is a method proposed by Hamilton et al. that is closely related to graph convolutional networks (GCN) [18]. Its name comes from "SAmple and aggreGatE". Unlike node2vec and other similar methods which rely on neighborhoods produced by random walks in order to generate one embedding per node, GraphSAGE works by using functions that aggregate information from nodes sampled from the local neighborhood of nodes. It is the aggregator functions themselves that are trained, not the vector representations. Because of this, the method is also capable of generating embeddings for previously unseen nodes. Also, GraphSAGE can leverage node attributes to generate embeddings. For example, in social networks these attributes could be gender, age, location, number of friends/followers, or any other information associated with the node. Structural features such as node degree can also be used as node attributes.

The algorithm

First, nodes are mapped to feature vectors. This step assumes that the parameters of K aggregator functions $AGGREGATE_k, \forall k \in 1, ..., K$, and K weight matrices $W^k, \forall k \in 1, ..., K$ have been learned. These functions are used to aggregate information from node neighbors and the weight matrices are used to propagate information in the graph. Nodes gain information from nodes that are further on the graph as the algorithm iterates over each depth.

At each depth k, for each node $v \in V$ an aggregated neighborhood vector $h^k_{N(v)}$ is calculated by

$$h^k_{N(v)} = AGGREGATE_k(h^{k-1}_u, \forall u \in N(v)) \tag{2.3}$$

where h^k_u denotes the feature vector of node u at step k, and $N(v)$ denotes the

neighborhood of node v. The resulting vector $h^k_{N(v)}$ has the same dimensionality as each vector h^{k-1}_u. Then h^k_v is calculated by

$$h^k_v = \sigma(W^k \cdot CONCAT(h^{k-1}_v, h^k_{N(v)})) \tag{2.4}$$

where σ is a non-linearity. The $CONCAT$ operation doubles the dimension of the vector, so the weight matrix W^k transforms the representation back to the intended size. At depth $k = 0$, h^0_v is defined as the attributes of node v. In summary, in order to generate the representation for a node at depth k, neighborhood representations at depth $k-1$ are aggregated and concatenated with the node's representation at depth $k-1$, and the resulting vector is weighted and fed through a fully connected layer with a nonlinear activation function.

In GraphSAGE, neighborhoods are fixed-size sets of neighbors that are sampled uniformly at each depth, in order to achieve a constant complexity for the processing of each batch. Several aggregator architectures can be used, such as mean, LSTM and pooling.

A graph-based loss function is applied to the output representations, and both the weight matrices and the parameters of the aggregator functions are tuned via stochastic gradient descent. The loss function can use node co-occurrence data from random walks on the graph, for example.

2.3.3 struc2vec

struc2vec is a method proposed by Ribeiro et al. that differs from other methods in what it encodes in its representations. While the previous methods optimize representations so that nearby nodes on the graph have similar representations, struc2vec optimizes representations so that nodes with similar structural roles, however far apart they are in the graph, have similar representations. The method uses auxiliary graphs that capture the structural similarities between the neighborhoods of each node at different distances. Also, since the objective is to capture structural similarity, node attributes are not used. See Figure 2.3 for a synthetic example of representations generated with struc2vec for the example network depicted in Figure 2.2. Note how representations of nodes that are structurally similar (e.g., D and G) are closer to each other than representations of nodes that are simply neighbors in the graph (e.g., D and A).

The algorithm

First, a measure of structural similarity is calculated for each pair of nodes in graph $G(V, E)$, considering neighborhoods at different distances. This structural distance metric between the k-hop neighborhoods of nodes u and v is given by

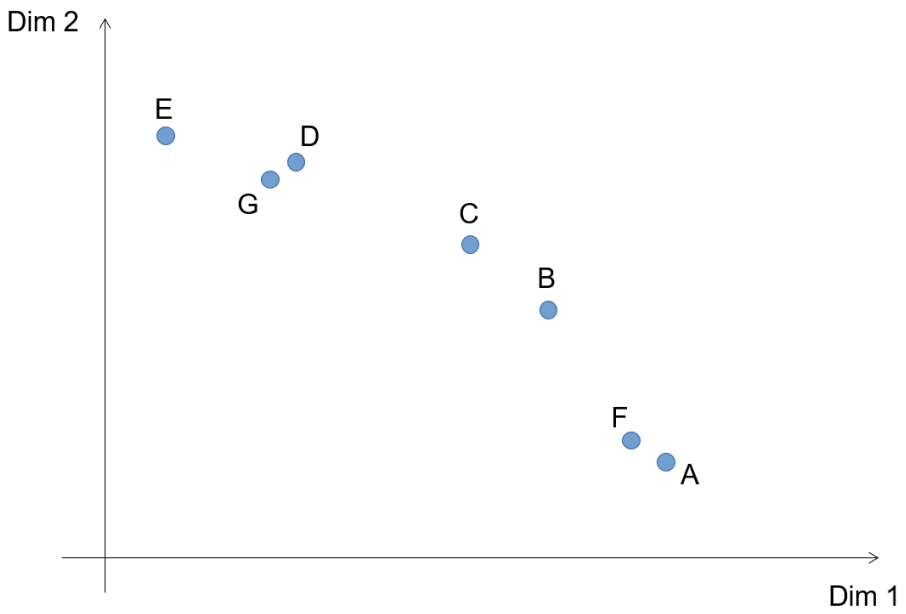


Figure 2.3: Synthetic example of struc2vec representations for the example network in Figure 2.2

$$f_k(u,v) = f_{k-1}(u,v) + g(s(R_k(u)), s(R_k(v))) \tag{2.5}$$

where $R_k(u)$ denotes the set of nodes at distance exactly k from u, $s(S)$ denotes the ordered degree sequence of a set $S \subset V$ of nodes, and $g(D1, D2) \geq 0$ measures the distance between the ordered degree sequences $D1$ and $D2$. struc2vec uses Dynamic Time Warping (DTW) for g, since $D1$ and $D2$ can be of different lengths. DTW requires a distance function between elements of two sequences in order to find the optimal alignment between the sequences, and the function chosen in struc2vec was $d(a,b) = \frac{\max(a,b)}{\min(a,b)} - 1$, where a and b represent the degrees of two nodes. Thus, for two identical sequences, the distance is equal to zero.

Then, a multi-layer graph M is constructed, where each layer corresponds to a hop distance (value for k). Each layer is a weighted undirected complete graph containing all nodes in the original graph, and edge weights within a layer are inversely proportional to the structural distance between each pair of nodes. Layers are connected with directed edges between corresponding nodes in each layer, and these edges are weighted such that a random walk is encouraged to traverse to an upper layer when the node is similar to many other nodes in the current layer, so that the node can be better differentiated from the others.

Multiple contexts are generated for each node using biased random walks of fixed length in the multi-layer graph, starting from each node's correspondent node in layer 0 of M. At each step of the walk, with probability q the walk stays in the current layer, and with probability $1 - q$ it changes layers. Since the weights

of edges between structurally similar nodes are higher, contexts will tend to have nodes that are structurally similar to the starting node, regardless of their distances in the original graph.

Finally, the Skip-gram model is used with Hierarchical Softmax in order to learn embeddings from the contexts that have been generated by the biased random walks in the multi-layer graph M. This last procedure to learn representations is very much like node2vec.

2.4 Embeddings Applied to Node Classification

Node classification is probably one of the most common applications for evaluating node embeddings with respect to their ability in capturing relevant information in low-dimensional representations. The goal is to accurately label the network nodes according to some characteristic that can be exogenous or endogenous, from the point of view of the network. Embeddings have been used to classify individuals [17], web pages [8], proteins [1, 2], and many other types of entities. The most common task is supervised classification, where node labels are available and used to train a classification model with the node embeddings as input. The accuracy of the trained model is then evaluated in a test set of the same graph.

node2vec [1] evaluates its representations on three different networks: a social network of bloggers, with labels that reflect the interests of bloggers as inferred from metadata; a protein-protein interaction (PPI) network, with labels representing biological states; and a co-occurrence network of words from Wikipedia, with labels representing part-of-speech tags also inferred from the data. The representations generated with node2vec showed varying levels of improvement in classification accuracy when compared to DeepWalk [8] and LINE [17] on all three networks.

GraphSAGE [2] also evaluates the performance of its representations on three networks: an academic paper citation network, with labels according to the subjects of the papers; a network of Reddit posts that belong to different communities; and a protein-protein interaction network, with labels according to protein function. The paper compares four variants of GraphSAGE against four baseline methods: a random classifier, a logistic regression classifier, the DeepWalk algorithm, and a combination of node features and DeepWalk embeddings. For all three networks, GraphSAGE outperformed all baselines (for the PPI network, DeepWalk was not evaluated, however).

struc2vec [3] evaluates its embeddings on three air traffic networks where nodes correspond to airports and edges correspond to the existence of commercial flights between two airports. Labels were assigned according to the level of activity of each airport, measured in number of people that arrived or departed each airport, or in the

total number of arrivals and departures. Four variants of the method were compared to node2vec embeddings and just the node degree as features, outperforming both on all networks.

One important discussion, however, concerns the shortcomings of existing evaluation strategies for semi-supervised node classification in graphs. A recent study [19] has shown that changes in training/validation/test data splits, training procedure and hyperparameters can lead to dramatic changes in the rankings of different methods. According to the study, many works have tested their proposed models on the same data splits of the same three datasets from a single paper, which favors the model that is most likely to overfit. It is also pointed out that considerably different training procedures are often used for each method, which leads to confusion as to whether improvements are due to the superiority of a method or a training procedure that is more adequate, with better tuned hyperparameters. The paper performs a thorough experimental investigation of these issues on four different Graph Neural Network (GNN) architectures (one of them being GraphSAGE) and four baseline models (not GNN-based) on four well-known citation networks and four newly introduced datasets. The study finds that GNN-based approaches outperform all baselines across all datasets. However, among the four GNN-based approaches, there was not a clear winner across all datasets, with GCN [18], a model that is simpler than the other three, being ranked best when relative accuracy (a metric that is described in detail in the paper) is considered. This indicates that simpler models can often outperform more sophisticated models when equally careful tuning is performed for all methods.

The study presented in Chapter 3 is orthogonal to such findings, as it characterizes the impact of the inherent randomness in these algorithms in the consistency of the embeddings that are generated. Indeed, it will be shown that such randomness can significantly influence the embeddings across different executions over the same data.

2.5 PageRank and Personalized PageRank

PageRank [4] is a method proposed by L. Page and S. Brin for ranking the relative importance of nodes in a directed graph according to the connectivity of nodes in terms of their degrees. Its origin is in Google's search engine, as it was designed to provide better search results based on a metric of importance for web pages. The core idea of PageRank is that web pages that are referenced by several other web pages (larger in-degree) are more relevant than web pages that are referenced by very few other pages. Also, if the pages that reference a certain web page are important, the importance of the page is also higher than that of a web page that

has a similar number of references, but that is only referenced by less relevant pages.

Formally, PageRank defines the rank of a node u as

$$R(u) = c \sum_{v \in B_u} \frac{R(v)}{N_v} \tag{2.6}$$

where B_u is the set of nodes that have a link to u (the in-neighborhood of u), N_v is the number of pages to which the node v links (the out-degree of v), and c is a normalizing factor that guarantees that the sum of the ranks of all nodes is constant.

To solve for the fixed point of equation 2.6, an iterative procedure can be used:

$$R_{t+1}(u) = c \sum_{v \in B_u} \frac{R_t(v)}{N_v} \tag{2.7}$$

This iterative process updates ranks until convergence. This can also be achieved with matrix multiplications, more specifically by multiplying the row vector of ranks $x_t = (R_t(u_1), R_t(u_2), \ldots, R_t(u_n))$ and a transition matrix A repeatedly, as in

$$x_{t+1} = x_t A \tag{2.8}$$

$A_{u,v}$is equal to $\frac{1}{N_u}$ if an edge exists from u to v, and equal to zero otherwise.

However, this means that nodes with no links to other nodes work essentially as "rank sinks", accumulating importance over time and never distributing it back to the rest of the nodes in the network. In order to address this limitation, PageRank introduces a vector S that corresponds to a source of rank. Ranks are then updated by the equation

$$R(u) = c\Big(\sum_{v \in B_u} \frac{R(v)}{N_v} + S(u) \Big) \tag{2.9}$$

Usually, S is just a vector of length $|V|$ in which every element has the same value, working essentially as a uniform distribution over the nodes, regardless of the edges. This allows rank information to be distributed back to the rest of the network, as sinks can be escaped.

Intuitively, the PageRank equations can also represent a random walker that traverses the web clicking on hyperlinks randomly. As the walk traverses edges, nodes that are well-connected will be visited much more often than nodes that are not, thus accumulating "importance" expressed by the number of times they have been visited. Mimicking the behavior of a person surfing on the web clicking links randomly, when a sink is reached, it is likely that the surfer will eventually just jump to some randomly chosen web page.

2.5.1 Personalized PageRank (PPR)

Even though its initial purpose was to address the problem of rank sinks, the vector S is useful for a different application. In contrast with its common form (i.e., a uniform distribution over all nodes), in the personalized form of PageRank it is defined differently, in order to focus the random choice on a subset of nodes, or even a single node (i.e., a target node). For example, instead of having a uniform distribution, S has all its elements equal to zero, except for the value of the node that corresponds to a web page of interest (e.g., Google's homepage), which is equal to one.

Under this definition for S, PageRank is defined in matrix form by

$$x_{t+1} = (1 - \alpha)x_t A + \alpha S \tag{2.10}$$

with α between 0 and 1, essentially works as a random walk that has a probability of $1 - \alpha$ of traversing an edge, and a probability of α of jumping back to the target node. As a consequence, the length of the walk before it returns to the target node follows a geometric distribution with parameter α, and thus has expected value $\frac{1}{\alpha}$, effectively limiting the number of steps taken by the walker before it returns to the target. This makes the algorithm biased towards nodes that are close to the target, thus visiting these nodes much more often than nodes that are farther from the target. In other words, the importance values assigned by PageRank are personalized to the target. As α approaches 1, the walk stays closer and closer to the target, without reaching further into the network, and thus the target node matters considerably. On the other hand, as α approaches zero, the influence of the target node becomes less and less important, since the walker will visit nodes over the entire network (assuming it is strongly connected).

Chapter 3

Empirical Analysis of Related Works

In this Chapter we present an empirical analysis of the embeddings generated by GraphSAGE and struc2vec. We use several similarity metrics in order to determine how consistent these methods are across executions, as both methods are non-deterministic and yield different results each time they are executed. We also evaluate how robust they are to the removal of nodes (along with their edges). The chosen metrics were Euclidean distance, the Jaccard similarity coefficient and Spearman's rank correlation coefficient.

3.1 Datasets and Methodology

This analysis was conducted on three undirected graphs: a Facebook egonet, a protein-protein interaction (PPI) network, and a graph of commercial flights between airports in the United States of America. The PPI network was obtained from GraphSAGE's public github page [20], while both the Facebook egonet and the USA airports networks were downloaded from struc2vec's public github page [21], although the egonet is also publicly available in the Stanford Large Network Dataset Collection [22] provided by SNAP (Stanford Network Analysis Project).

As Table 3.1 shows, these networks are very different, with the largest having almost 66 times the number of nodes of the smallest, and almost 72 times the number of edges. The average node degree is somewhat similar across all networks, even though the median degree varies considerably. The average clustering coefficient is much lower in the PPI network, which also has several connected components, unlike the Facebook egonet, which has a single connected component. Even though the Facebook egonet is the smallest network in number of nodes, it has the largest diameter. In contrast, its average shortest path length is the smallest.

Table 3.1: Summary metrics of chosen networks

Metric	Facebook egonet	PPI	USA airports
Number of nodes	224	14755	1190
Number of edges	3192	228431	13599
Minimum degree	1	1	1
Maximum degree	99	722	238
Average degree	28.50	30.96	22.86
Median degree	22	17	6
Average clustering coefficient	0.54	0.18	0.50
Number of connected components (CC)	1	68	3
Size of largest CC	224	3312	1186
Diameter of largest CC	9	7	8
Average shortest path length of largest CC	2.52	2.77	3.07

For each network, induced subgraphs were randomly generated, each one containing a percentage of the nodes in the original graph. These percentages range from 10% to 90% in steps of 10%. In order to choose the nodes of the induced subgraph, a random permutation of the nodes was generated, and the first P nodes are used, where P is the number that represents the corresponding percentage of nodes in the network. Node embeddings were generated with both methods (GraphSAGE and struc2vec) independently for each graph and induced subgraph.

Since we have two independent sets of embeddings for each graph and subgraph, we measured how much the node representations vary from one execution to another, at all levels of node removal and including the original network. Also, since we have embeddings for different levels of node removals, we studied how the representations are affected as nodes and edges are removed from the graph.

We adopt $x_u^{i,M}$ as the notation for the resulting vector representation (embedding) for node u on the i-th execution of method M. However, when comparing only embeddings generated with the same method, the method indicator (M) will be dropped from the superscript for the sake of simplicity (i.e., $x_u^{i,M}$ becomes x_u^i). Also, two distance functions between two node representations will be used: $d^{i,j}(u)$ and $d^i(u, v)$. The former compares embeddings generated for the same node u in two different executions i and j of a method, while the latter compares embeddings generated for two different nodes u and v in the same execution i of a method. Finally, vectors and sequences will be indexed using the notation $x[i]$ for the i-th element of vector or sequence x, starting from 1.

3.2 Euclidean distance

The euclidean distance between two vectors x and y in n-dimensional space is given by

$$
\begin{aligned}
d(u, v) &= ||x_u - x_v|| \\
&= \sqrt{(x_u[1] - x_v[1])^2 + (x_u[2] - x_v[2])^2 + ... + (x_u[n] - x_v[n])^2}
\end{aligned}
\tag{3.1}
$$

Euclidean distances were used in two different ways: distance between corresponding representations, and difference between two corresponding distances. These metrics are more clearly described in what follows.

3.2.1 Distance between corresponding representations

Given two sets of embeddings, for every node in the intersection of their node sets we calculate the distance between the representations that were generated for the node. For node u in the intersection, we calculate the metric

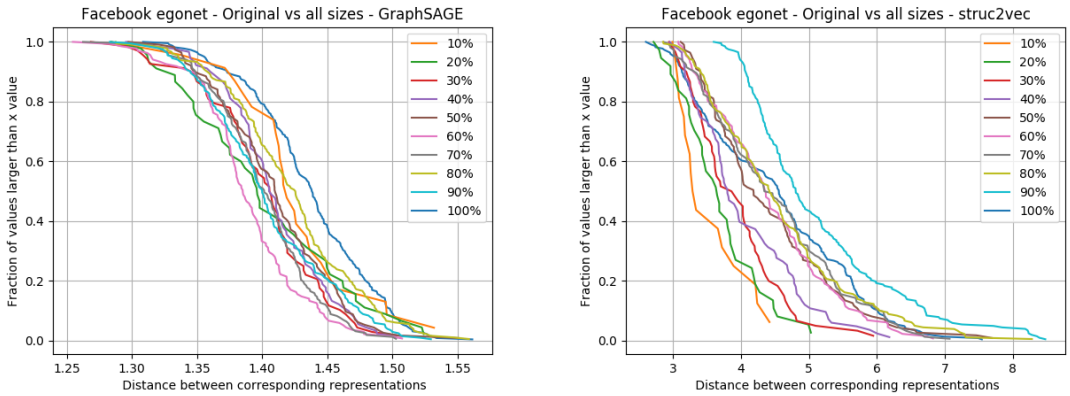
$$
d^{1,2}(u) = ||x_u^1 - x_u^2|| \tag{3.2}
$$

where x_u^i is the vector representation for node u originated from execution i of a method.

These distances are used to plot the complementary cumulative distribution function (CCDF). On a CCDF, the x-axis contains values of some measurement, and the y-axis contains, for each value in the x-axis, the fraction of values that are larger than or equal to that value. For example, if the curve passes through the point $(10, 0.5)$, this means that 50% of values in the distribution are larger than or equal to 10. Due to its nature, a CCDF is always monotonically decreasing.

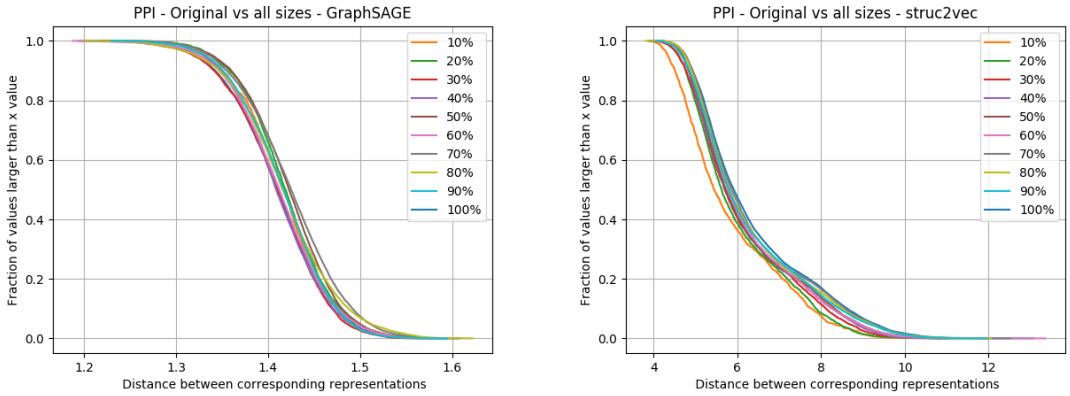
When the curve of a CCDF distribution decays slowly, it means that there are few values on that interval of values in the x-axis, and when it decays rapidly, it means that there are many values on that interval. For example, on Figure 3.2a, every depicted curve decays slowly before 1.3 and after 1.55, meaning there are very few values on those intervals, but it decays rapidly between 1.3 and 1.55, meaning most values belong to that interval. Note that this curve shape indicates a normal distribution of distance values, as it is almost symmetrical in the x-axis and most values are concentrated around a mean, with smooth decay.

Figures 3.1, 3.2, and 3.3 show the CCDF distributions of distances between corresponding representations, for all three networks. The first observation from these Figures is that the space to which GraphSAGE representations belong is more consistent than the one to which struc2vec representations belong, since GraphSAGE distances do not vary much in magnitude from one network to another (i.e., domain of distances in x-axis). However, neither method is very consistent from one execution to another, because the average distance between corresponding representations



(a) GraphSAGE (b) struc2vec

Figure 3.1: CCDF of distances between corresponding representations (Facebook egonet — Original vs all sizes)

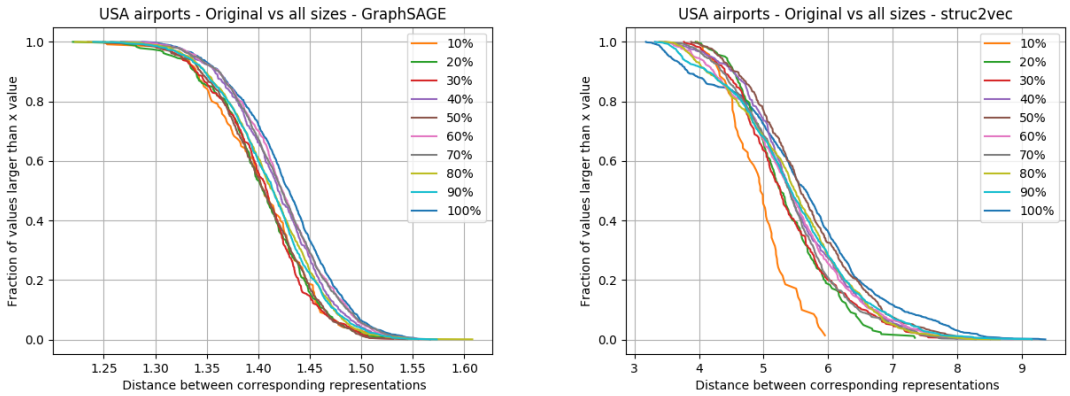


(a) GraphSAGE (b) struc2vec

Figure 3.2: CCDF of distances between corresponding representations (PPI — Original vs all sizes)

is much larger than zero. This is expected, since both methods have a considerable amount of randomness, which means each execution can create a completely different vector representation for the nodes.

Note that network structure and size play an important role on the behavior of the representations. The curves for the Facebook egonet are much less concentrated than those of the USA airports network, and the curves for the PPI network are even more concentrated. Also note that curves for different levels of node removal do not seem to follow an ordering. Distances when comparing the original network to the network after only 10% of the nodes have been kept might be the smallest among all scenarios (see the 10% curve on Figure 3.3b), or they might be among the largest (see the 10% curve on Figure 3.1a).



(a) GraphSAGE

(b) struc2vec

Figure 3.3: CCDF of distances between corresponding representations (USA airports — Original vs all sizes)

3.2.2 Difference between corresponding distances

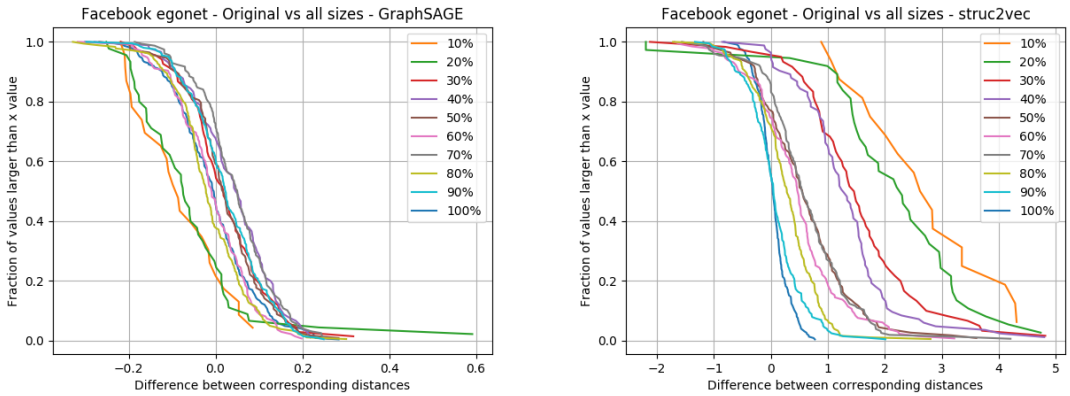
Given two sets of embeddings (e.g., for a graph and an induced subgraph), for every node in the intersection of their node sets we calculate the distance between the representation that was generated for a node and the representation that was generated for another (randomly sampled) node, in each embedding. We then calculate the difference between these distances. So for a node u, we first sample a node v uniformly and calculate both $d^1(u,v) = ||x_u^1 - x_v^1||$ and $d^2(u,v) = ||x_u^2 - x_v^2||$. Then, we calculate the difference as defined by

$$g(u,v) = d^1(u,v) - d^2(u,v) \tag{3.3}$$

Once again, these values are used to plot a complementary cumulative distribution function, indicating the fraction of distance differences that are larger than a given value.

Table 3.2 shows the average, standard deviation and median values of differences between corresponding distances on all three networks, with GraphSAGE and struc2vec. Both GraphSAGE's and struc2vec's differences between corresponding distances have an average value that is close to zero, which is a sign of consistency when it comes to the relative positions of the representations of different nodes from one execution to another for the same graph.

When comparing Figures 3.4, 3.5, and 3.6, it also becomes evident that, in general, GraphSAGE does a better job at keeping the average close to zero as an increasing percentage of nodes and edges is removed from the graph. In other words, the distribution curves are more tightly packed for GraphSAGE than they are for struc2vec. Note that with struc2vec the curves of all three networks follow an almost strict ordering, with curves of lower percentages of nodes kept on the network aver-



(a) GraphSAGE

(b) struc2vec

Figure 3.4: CCDF of difference between corresponding distances (Facebook egonet — Original vs all sizes)

aging at higher values. This is to be expected, since struc2vec is highly dependent of the structural roles of nodes, and the more nodes and edges are removed, the larger the difference in the structure around each remaining node. However, this happens for the USA airports network with GraphSAGE as well, most likely due to the fact that network structure and size also seem to play an important role in the behavior of the distributions, as the behavior of each network is considerably different from those of the other two networks when comparing the same embedding method.

Table 3.2: Summary metrics of difference between corresponding distances (Original vs copy)

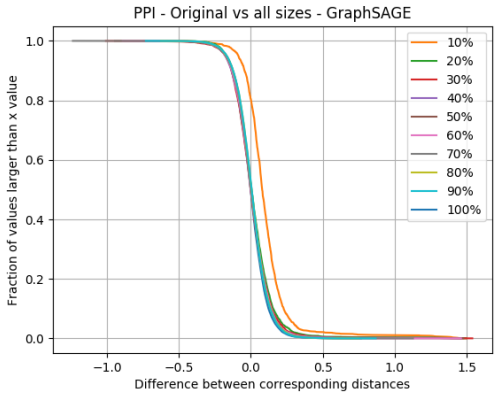
Graph	Method	Average ($*10^{-3}$)	Std. Deviation ($*10^{-1}$)	Median ($*10^{-3}$)
Facebook egonet	GraphSAGE	-6.67	0.98	-4.08
Facebook egonet	struc2vec	29.51	2.77	30.50
PPI	GraphSAGE	-0.51	1.06	-0.89
PPI	struc2vec	6.70	2.70	3.18
USA airports	GraphSAGE	9.98	1.02	9.99
USA airports	struc2vec	-9.53	2.61	-7.59

3.3 Jaccard similarity coefficient

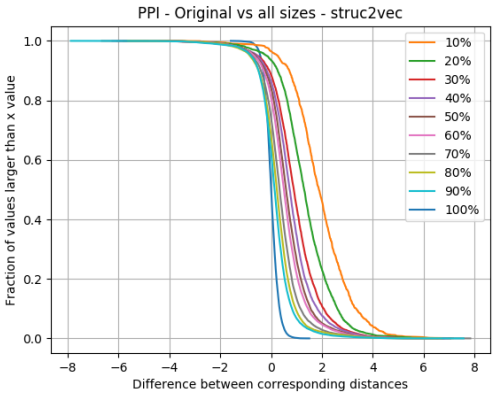
The Jaccard similarity coefficient measures the similarity between two finite sets, and is simply the ratio between the sizes of the intersection and the union of these sets. So, the Jaccard similarity coefficient for two sets A and B is given by

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.4}$$

This is a very direct and intuitive way to measure the similarity between two sets when the order of elements is not relevant, since the more elements in common

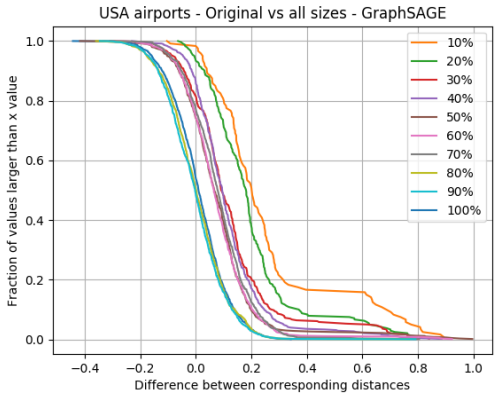


(a) GraphSAGE

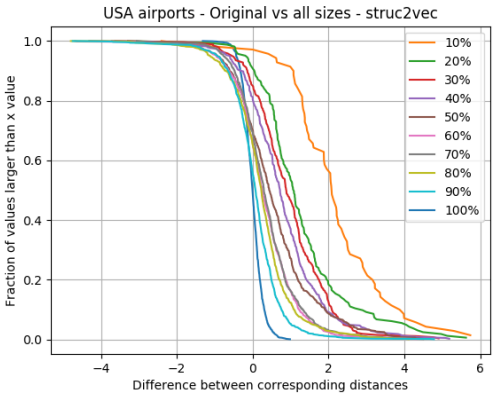


(b) struc2vec

Figure 3.5: CCDF of difference between corresponding distances (PPI — Original vs all sizes)



(a) GraphSAGE



(b) struc2vec

Figure 3.6: CCDF of difference between corresponding distances (USA airports — Original vs all sizes)

the sets have, the closer to 1 is the result, and the fewer elements in common, the closer to 0 it gets. However, the interpretation of results can be somewhat confusing, because the denominator decreases as the numerator increases, leading to nonlinear growth of the coefficient. Particularly, for two sets of same size N, if $n = |A \cap B|$, then the growth of the coefficient can be described by the function $f(n) = \frac{n}{2N-n}, 0 \leq n \leq N, n \in \mathbb{N}$.

We use the Jaccard coefficient to measure the similarity between the top-k closest representations for each node in the two sets of embeddings. So, for node u with representation x_u^1 generated from the first execution of a method, the euclidean distance between x_u^1 and the representation x_v^1 for every node v in the graph is calculated, and the label of the k closest nodes are selected for the set $S_k^1(u)$. That is,

$$S_k^1(u) = \{v | v \in V - \{u\}, \text{ and } d^1(u, v) \leq D^1(u)[k]\} \tag{3.5}$$

where

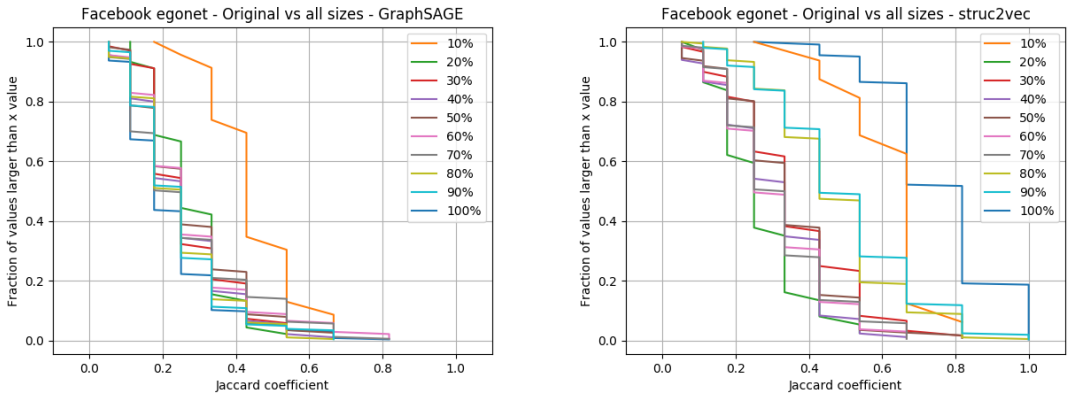
$$D^1(u) = \{d^1(u, v) | v \in V - \{u\}\} \tag{3.6}$$

is sorted in ascending order. This is also done for the second execution. $S_k^1(u)$ and $S_k^2(u)$ are different because both GraphSAGE and struc2vec are non-deterministic and lead to different embeddings each time they are executed. Then the Jaccard coefficient

$$J(S_k^1(u), S_k^2(u)) = \frac{|S_k^1(u) \cap S_k^2(u)|}{|S_k^1(u) \cup S_k^2(u)|} \tag{3.7}$$

is calculated for these two sets. This process is repeated for every node in the graph. Finally, these similarity coefficients are used to generate a CCDF plot. In our analysis, we ran this process with $k = 10$ (top-10), $k = 100$ (top-100) and $k = \frac{|V|}{10}$ (top-10%) closest representations.

It is important to note that when the number of nodes in the network becomes too small, the distributions are artificially shifted to a higher average, since it is easier to keep the same nodes in the top-k when there are less nodes to choose from. In other words, it is much easier to keep the same 10 nodes in the top-10 when the network has 20 nodes than it is when it has 200 nodes. This becomes clear when observing the distribution of Jaccard coefficients for the network after removing 90% of the nodes (see Figures 3.7, 3.10, and 3.12). Because of this issue, we have left top-100 results out of the text since it suffers too much from this problem and conveys little meaning.



(a) GraphSAGE

(b) struc2vec

Figure 3.7: CCDF of top-10 Jaccard coefficients (Facebook egonet — Original vs all sizes)

3.3.1 Jaccard top-10

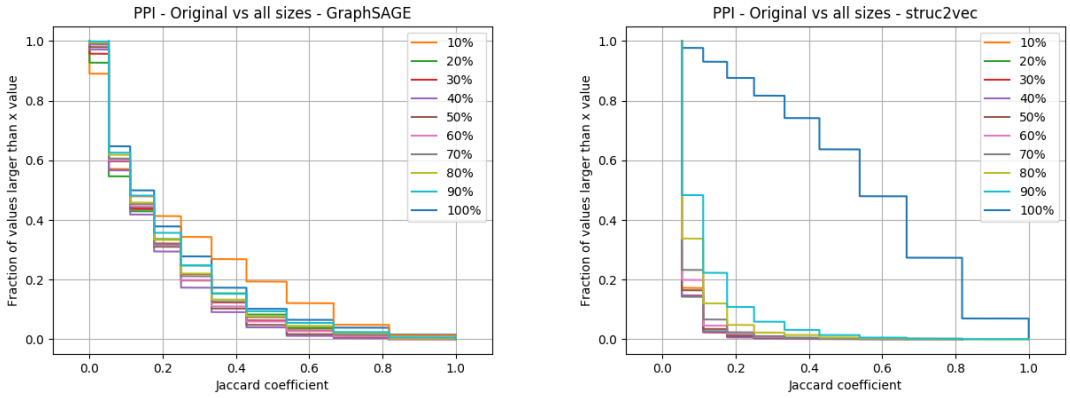
Table 3.3 shows the average, the standard deviation and the median values of Jaccard coefficients for the top-10 closest nodes on the same network, for all three networks, with GraphSAGE and struc2vec. The averages with struc2vec are considerably closer to 1 than with GraphSAGE, which are much closer to 0. This shows that struc2vec does a much better job at keeping the representations of the same nodes close together across different executions with the same network. In contrast, Figures 3.7, 3.8, 3.9 show that struc2vec suffers severely when nodes and edges are removed. When too many nodes are removed, the distribution in struc2vec resembles that of GraphSAGE.

Table 3.3: Summary metrics of top-10 Jaccard coefficients (Original vs copy)

Graph	Method	Average ($*10^{-1}$)	Std. Deviation ($*10^{-1}$)	Median ($*10^{-1}$)
Facebook egonet	GraphSAGE	2.21	1.35	1.76
Facebook egonet	struc2vec	7.55	1.60	8.18
PPI	GraphSAGE	2.19	2.08	1.11
PPI	struc2vec	5.59	2.53	5.38
USA airports	GraphSAGE	1.26	1.52	0.53
USA airports	struc2vec	7.22	2.19	8.18

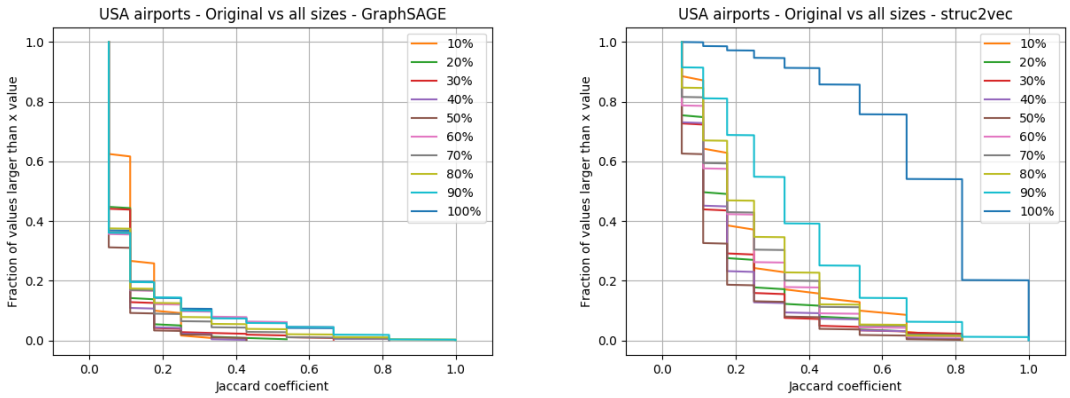
3.3.2 Jaccard top-10%

Table 3.4 shows the average, the standard deviation and the median values of Jaccard coefficients for the top-10% closest nodes on the same network, for all three networks, with GraphSAGE and struc2vec. The behavior of top-10% distributions is similar to the top-10 scenario. struc2vec does a better job than GraphSAGE at keeping the same nodes close together, while suffering more from the removal of nodes. There is,



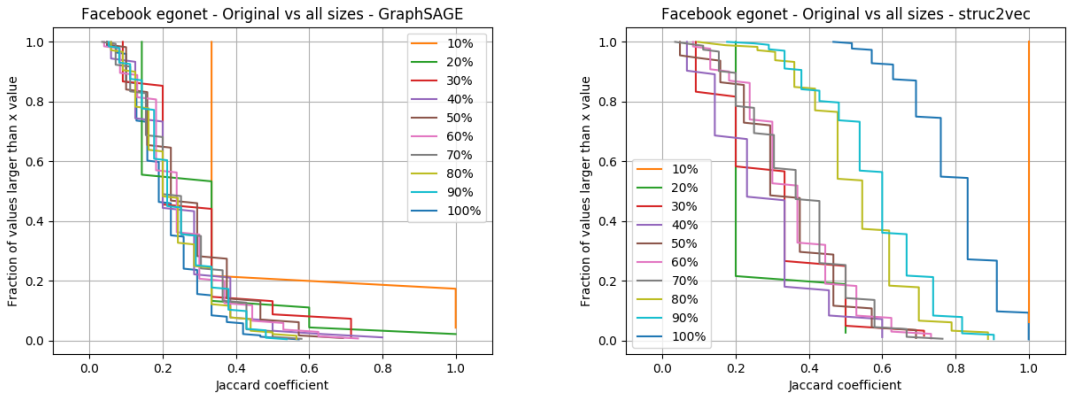
(a) GraphSAGE (b) struc2vec

Figure 3.8: CCDF of top-10 Jaccard coefficients (PPI — Original vs all sizes)



(a) GraphSAGE (b) struc2vec

Figure 3.9: CCDF of top-10 Jaccard coefficients (USA airports — Original vs all sizes)



(a) GraphSAGE

(b) struc2vec

Figure 3.10: CCDF of top-10% Jaccard coefficients (Facebook egonet — Original vs all sizes)

however, some change in the values of averages, the biggest ones being for the PPI network with struc2vec (34.70% lower average on top-10% scenario) and the USA airports network with GraphSAGE (33,33% lower average on top-10% scenario).

Table 3.4: Summary metrics of top-10% Jaccard coefficients (Original vs copy)

Graph	Method	Average ($*10^{-1}$)	Std. Deviation ($*10^{-1}$)	Median ($*10^{-1}$)
Facebook egonet	GraphSAGE	2.13	0.96	1.89
Facebook egonet	struc2vec	7.98	1.20	8.33
PPI	GraphSAGE	1.96	0.79	1.85
PPI	struc2vec	3.65	1.00	3.92
USA airports	GraphSAGE	0.84	0.36	0.77
USA airports	struc2vec	6.32	1.79	5.97

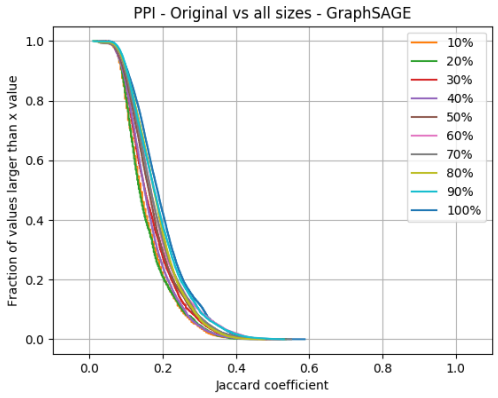
3.4 Spearman's rank correlation coefficient

Spearman's rank correlation coefficient measures the statistical dependence between the rankings of two variables. Unlike Pearson correlation coefficient, which measures linear correlation and only results in a perfect score if the relationship is perfectly linear, Spearman's coefficient assesses how well the relationship between two variables can be described by a monotonic function, be it linear or not. It is defined as

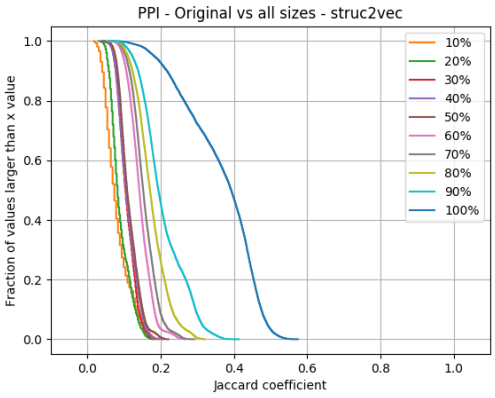
$$s(x, y) = \frac{cov(r_X, r_Y)}{\sigma_{r_X} \sigma_{r_Y}} \tag{3.8}$$

where $cov(x, y)$ is the covariance between variables x and y, σ_X is the standard deviation of variable X, and r_X is the rank variable of X [23].

We use Spearman's correlation coefficient to assess the relationship between the distances from a specific node to all nodes in the graph, for two different executions of

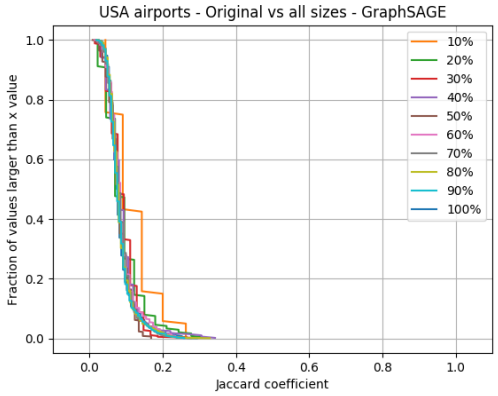


(a) GraphSAGE

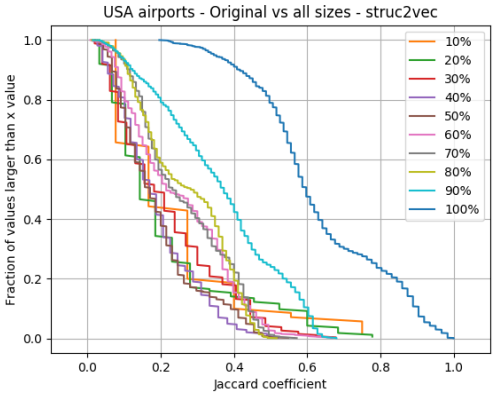


(b) struc2vec

Figure 3.11: CCDF of top-10% Jaccard coefficients (PPI — Original vs all sizes)



(a) GraphSAGE



(b) struc2vec

Figure 3.12: CCDF of top-10% Jaccard coefficients (USA airports — Original vs all sizes)

an embedding method. So in our case X and Y are, respectively, a list of distances $d^1(u, v)$ and a list of distances $d^2(u, v)$, with node u fixed in both cases. If the distance from node u to each other node v does not change considerably from one execution to another, the ranks will be very similar, and the coefficient will be close to 1.

We run this for every node in the graph, which results in a list of values of Spearman's correlation coefficient for the whole graph. These values are used to generate a CCDF plot.

Table 3.5: Summary metrics of Spearman coefficients (Original vs copy)

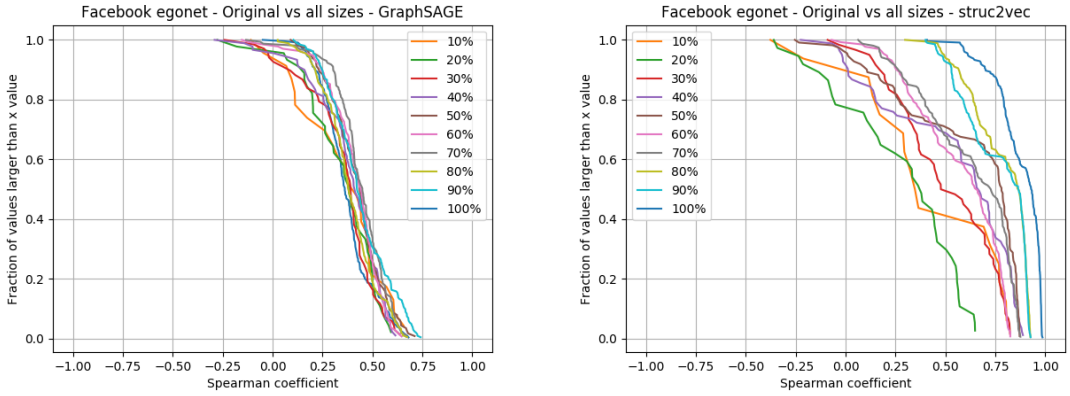
Graph	Method	Average ($*10^{-1}$)	Std. Deviation ($*10^{-1}$)	Median ($*10^{-1}$)
Facebook egonet	GraphSAGE	3.71	1.29	3.61
Facebook egonet	struc2vec	8.75	1.18	9.29
PPI	GraphSAGE	2.80	1.20	2.86
PPI	struc2vec	9.04	0.54	9.26
USA airports	GraphSAGE	0.65	0.54	0.63
USA airports	struc2vec	8.73	0.71	8.88

Table 3.5 shows that for two independent sets of vector representations for the same graph, the Spearman coefficient averages are much higher for struc2vec than for GraphSAGE. This means that, for each node, the order of nodes from least distant to most distant is considerably better preserved with struc2vec.

Figures 3.13, 3.14 and 3.15 show the CCDF of Spearman coefficients for GraphSAGE and struc2vec for the three networks. Note that the different curves for struc2vec follow an almost strict ordering with larger values associated with larger node overlap. This is particularly striking for the USA airports network (Figure 3.15b). Clearly, the removal of nodes strongly influences the distribution of the Spearman coefficients in struc2vec, due to the fact that structural information is lost rapidly when this is done. In contrast, the Spearman distributions for GraphSAGE show no clear ordering with respect to node removal. Moreover, GraphSAGE distributions are significantly smaller in comparison to struc2vec, and in particular for the case in which the network is identical (100% curve). On the USA airports network, this is especially striking. Finally, except for a few cases in which most of the network's nodes have been removed, struc2vec's Spearman coefficient distributions are consistently ahead of GraphSAGE's distributions, which reinforces that struc2vec does a better job at preserving the order of nodes from least to most distant.

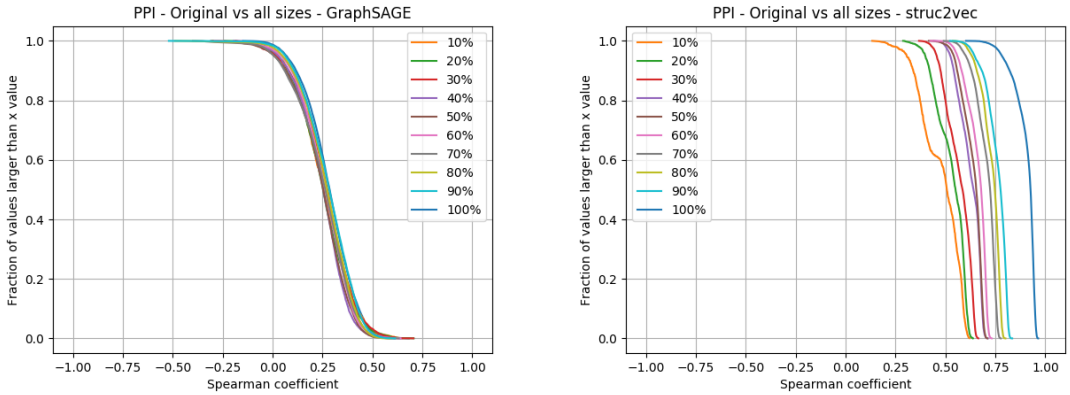
3.5 Conclusions

In summary, a few conclusions have been made:



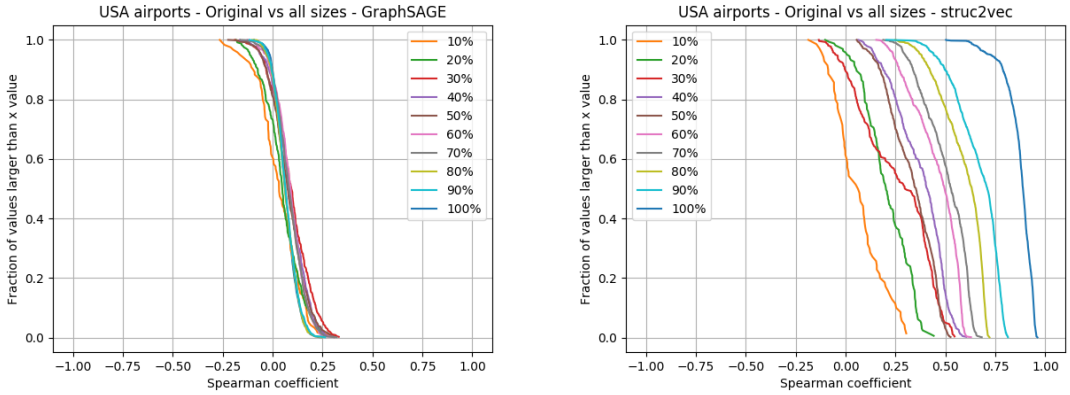
(a) GraphSAGE (b) struc2vec

Figure 3.13: CCDF of Spearman coefficients (Facebook egonet — Original vs all sizes)



(a) GraphSAGE (b) struc2vec

Figure 3.14: CCDF of Spearman coefficients (PPI — Original vs all sizes)



(a) GraphSAGE (b) struc2vec

Figure 3.15: CCDF of Spearman coefficients (USA airports — Original vs all sizes)

- Neither method's vector representations are very consistent from one execution to another for a network, since both methods have a considerable amount of randomness
- struc2vec suffers more when nodes and edges are removed, because it is highly dependent of the structural roles of nodes
- struc2vec does a better job at keeping the representations of the same set of nodes close together across different executions for the same network, as its Jaccard coefficient averages are higher than GraphSAGE's
- struc2vec does a better job at keeping the order of nodes from least distant to most distant, for each node, as its Spearman coefficient averages are higher than GraphSAGE's

Chapter 4

PPR Representations

Recall the iterative algorithm to compute the PageRank of nodes of a graph. This iterative process essentially generates a sequence of values, even though only the final value is commonly used. However, it is sensible to argue that other values in this sequence also hold valuable information concerning the structure around the node. Two nodes might end up having very similar values after convergence, but their journey there can be considerably different, depending on their local network structure, how many nodes they connect to, and the importance of the nodes to which they are connected.

Moreover, since these values are the result of a process that does not take into account neither the identity of nodes nor other attributes that are external to the structure of the graph, this measurement of importance is purely structural. Also, given a fixed set of initial conditions (i.e., the transition matrix, the initial value of each node and the vector S), the process is deterministic, since there are absolutely no random factors. This means independent executions over a certain graph, given the same initial conditions, will lead to identical results, every time.

Recall that the vector S can be used to generate levels of importance that are personalized for a single target node. The level to which the results are personalized depends on the value of the parameter α, which controls how far (in hops) information is expected to reach from the target node. As a consequence, the larger the value of α is, the more personalized the PageRank values are. By using different values for α, one can generate sequences with different levels of personalization.

Thus, the sequence of values for a node can be used as a "signature" for its structural role in the network, since two different nodes will have the same exact sequence if they are structurally identical, i.e. belong to the same automorphism group. Figure 4.2 shows the PageRank values at each iteration for all nodes of the network shown in Figure 4.1, for $\alpha = 0.01$. Note that after the first iteration, all nodes have the same PageRank value of 0.01 (i.e., equal to α). However, at the second iteration, PageRank values become different for different nodes. The excep-

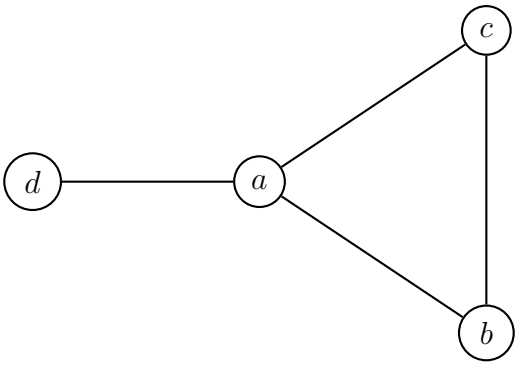


Figure 4.1: Example network for PPR

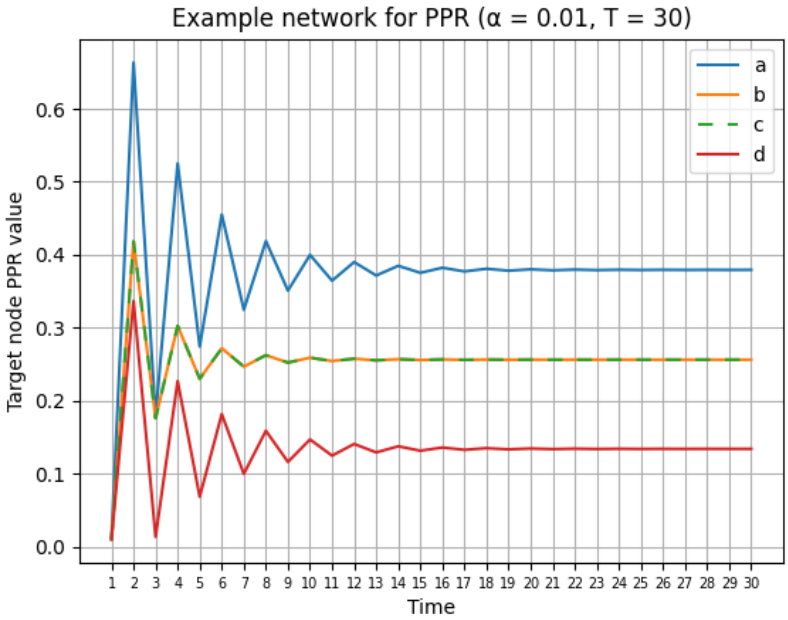


Figure 4.2: Evolution of target node PPR values over time for all nodes of the example network

tion are nodes b and c, whose sequences are identical due to them being structurally identical.

Also note on Figure 4.2 how the sequences clearly differentiate the nodes by their structural role. In addition, note that the PageRank values converge for all nodes after a large enough number of iterations. Moreover, different values for α lead to different sequences, meaning one can generate signatures for increasingly larger neighborhoods around the target node. This is the basis of the framework proposed for generating representations of nodes. Note that the relative importance of other nodes with respect to a target node is not used here. Instead, we are exclusively interested in the PageRank value for the target node as it evolves over the iterations.

4.1 The framework

Our framework has two main parameters: the return rate α and the horizon T, which controls the number of iterations for which PageRank values are generated. Running PageRank until convergence is not strictly necessary as the further we go the less the values vary from one iteration to another, so less relevant information is acquired with each new iteration. Limiting the number of iterations makes it possible to reduce the time required to generate the representations.

First we choose a set of values for α and a single value for T. Then, for each value of α we run the PPR algorithm for T iterations. Recall that x_t is the vector representing the Personalized PageRank values for all network nodes after t iterations, starting from a given target node. Note that only the entry corresponding to the target node is used to form its representation. Thus, the sequence of vectors x_t becomes a sequences of numbers, as shown in Figure 4.2.

The vector of initial node values is composed of all zeros, except for the position corresponding to the target node, which contains a 1. The same is true for the vector S. The transition matrix is such that transitions from each node can only occur along edges of the graph, uniformly. The output is a matrix representation for a given node, in which there is one row per value of α and one column per iteration.

It is important to note that the process runs independently for each node. This means that if the goal is to generate a representation for a single node, it is not necessary to generate representations for every node in the graph. Many of the other methods require that representations are generated for the entire network in order to acquire the representation of a specific node (e.g., [1, 3, 8]). However, this does not mean that the entire process has to run again for each node. Instead, for a given value of α, the iterative algorithm to compute PageRank can run once and generate representations for all nodes in the network.

Recall from equation 2.10 that the iterative procedure requires multiplying the vector x_t of values at step t by the transition matrix A, which results in another vector, and adding that to the vector S (after multiplying by $1-\alpha$ and α, respectively) in order to obtain x_{t+1}. While this is for a single node, the iterative procedure can also compute the PageRank values for all nodes at once if instead we multiply a matrix X_t, in which each row contains the row vector x_t of the respective target node, by the transition matrix A and add the result to a matrix M_S, in which each row contains the vector S of the respective target node. In other words, according to

$$X_{t+1} = (1-\alpha)X_tA + \alpha M_S \tag{4.1}$$

Perhaps an example will better illustrate the framework. Consider again the

network in Figure 4.1.

For this network, we have the transition matrix A as defined by

$$A = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.2}$$

In each row, we have the probability of transitioning from a node to every other node, as defined by the neighbors of the node.

If we were to generate a representation only for node a, we would have x_0 and S as defined by

$$x_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}$$

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.4}$$

In contrast, if we were to generate a representation for every node, we would have X_0 and M_S both be identity matrices, as defined by

$$X_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

$$M_S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

The transition matrix A stays the same. However, this can be done for any subset of nodes. Say we would like to generate representations only for nodes b and c. Then we would have X_0 and M_S as defined by

$$X_0 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.7}$$

$$M_S = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.8}$$

For a subset of k nodes, each iteration multiplies a (k x $|V|$) matrix by a ($|V|$ x $|V|$) matrix, an operation that has $O(k|V|^2)$ running time complexity. In the worst case scenario, in which $k = |V|$ (i.e., generating representations for every node in the

graph), each iteration has complexity $O(|V|^3)$. This leads to an overall complexity of $O(kT|V|^2)$, $1 \leq k \leq |V|$, when considering T iterations. Note that the sequence of PageRank values can possibly be computed by more efficient algorithms, which is instrumental for larger networks. However, this is outside the scope of this work.

4.2 Visualizing the representations

In the previous Section we saw an example in which two nodes that belong to the same automorphism group had the same PPR sequence for a chosen value for α. However, it was a small example with only four nodes and little diversity, with a single value for α. Let us now better visualize the method's ability to capture the structural role of nodes.

Consider a barbell graph $B(m_1, m_2)$ in which there are two complete graphs of size m_1 connected to each other by a line graph of size m_2. One node of each complete graph acts as a bridge by connecting to the line graph. The barbell graph is interesting for our application because it has many nodes that belong to the same automorphism group, as well as several nodes that are similar to one another, but not in the same automorphism group. All nodes that belong to one of the complete subgraphs are structurally identical, except for the ones that act as bridges, which are only structurally identical to one another. All of the nodes in the line subgraph have a mirror node that belong to the same automorphism group as them, and are structurally similar to their neighbors in the line subgraph. See Figure 4.3 for a barbell graph $B(10, 10)$ in which nodes are colored according to their structural role. Nodes 0-8 and 21-29 all belong to the same automorphism group. The same is true for the pairs (9, 20), (10, 19) and so on until (14, 15).

If our method is capable of capturing these structural roles, then the expected behavior is for this to be easily observed for the barbell graph. Also, it is expected that the representations for nodes that are not structurally identical, but still similar, are closer to each other than they are to the representations for nodes that are considerably different. For example, nodes 10 and 11 are much closer than nodes 10 and 14, from a structural point of view.

In order to see this in action, we generated PPR representations for all nodes in the barbell graph in Figure 4.3, with $T = 20$ and several values of α. Then, we reduced the dimensionality of the representations using Principal Component Analysis (PCA), resulting in 2-dimensional vectors. Finally, we generated a scatter plot with these vectors, coloring the representation of each node in the barbell graph with the same color used for the node in Figure 4.3. See Figure 4.4 for the results.

The first thing to be pointed out is that, as expected, structurally identical nodes have exactly the same representation, so each color only appears once. Secondly,

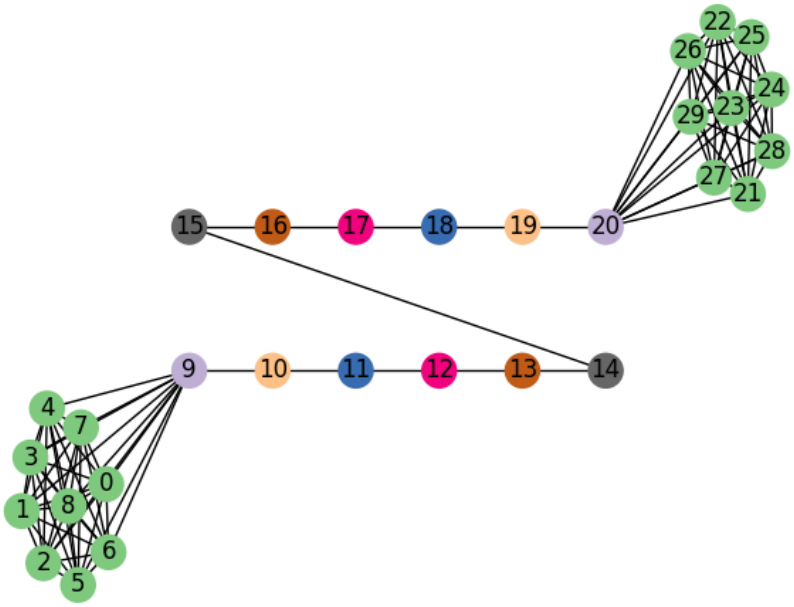


Figure 4.3: The barbell network $B(10, 10)$ with nodes colored by structural role

the method was able to identify and differentiate all roles in the graph. There are seven different structural roles, and seven clearly distinct points in the plot. Finally, it is clear that the representations also capture the structural similarity between the nodes. The points that refer to nodes in the center of the line subgraph are closer together, and the distance grows as nodes approach the complete subgraphs. Nodes that belong to the complete subgraphs and act as bridges to the line subgraph are also much closer to the other nodes in the complete subgraphs than they are to any of the nodes that only belong to the line subgraph.

4.3 Sensitivity Analysis on real networks

Figures 4.5, 4.6 and 4.7 show the evolution of the target node PPR value for all nodes in three different networks, respectively. In all three, each curve refers to a node in the network, and represents the evolution of the node's PPR value over time. It is evident that the PPR sequences of the nodes follow diverse patterns, and even nodes with similarly shaped sequences often converge to different values. Also, nodes that do converge to similar values often have considerably differently shaped sequences, which shows the relevance of the sequence itself, and not only its final value. However, these observations do not hold true for every value of α, as we will see soon.

Secondly, in order to better understand how the framework behaves for different networks, we also analyzed the influence of the parameters α and T on the final

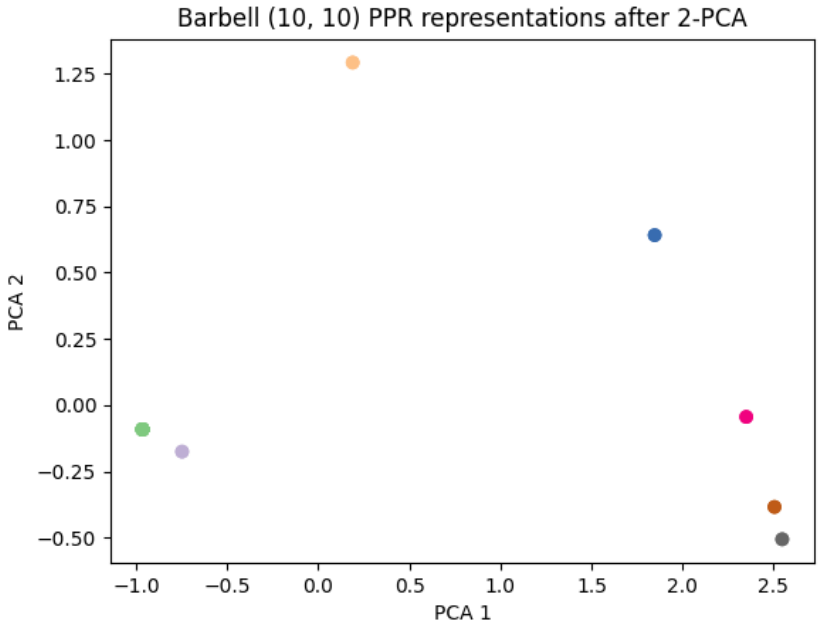


Figure 4.4: PPR representations of nodes in $B(10, 10)$ after 2-PCA, with colors corresponding to each structural role

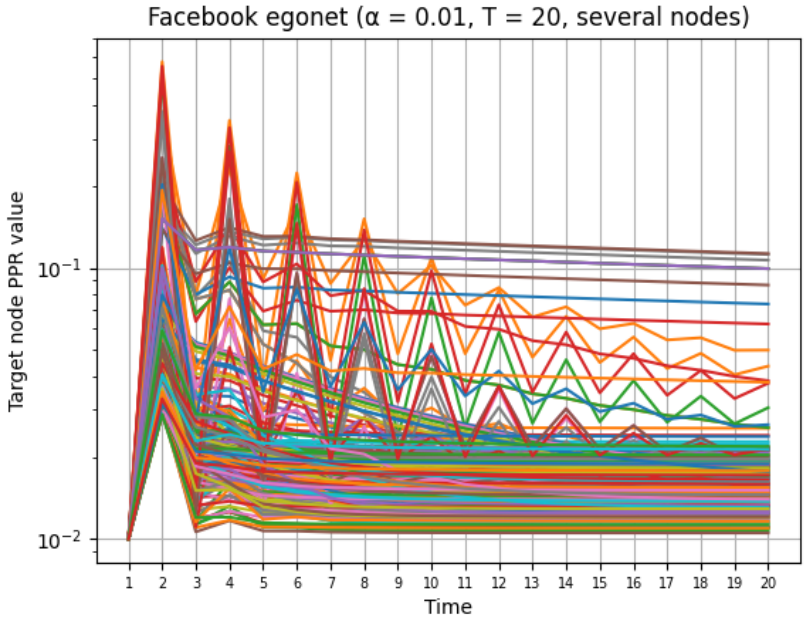


Figure 4.5: Evolution of target node PPR values over time for all nodes of the Facebook egonet

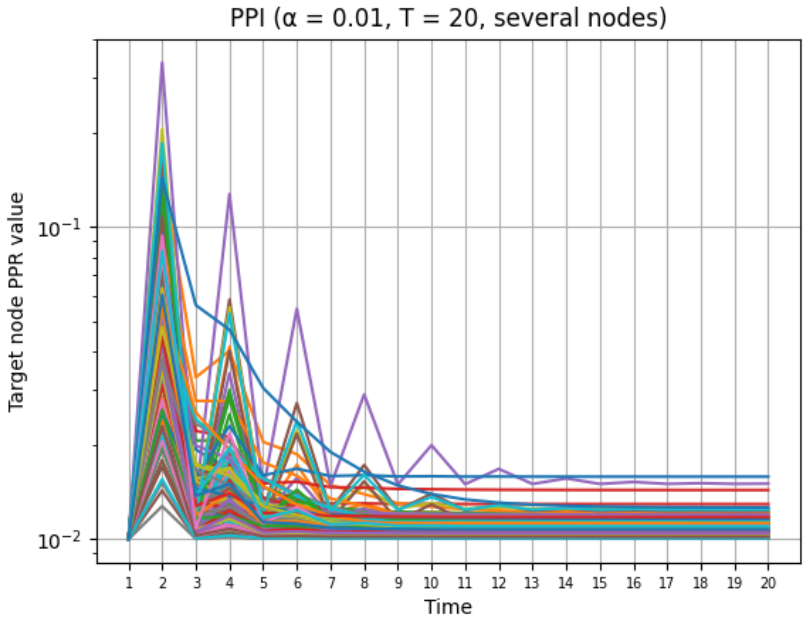


Figure 4.6: Evolution of target node PPR values over time for several nodes of the PPI network

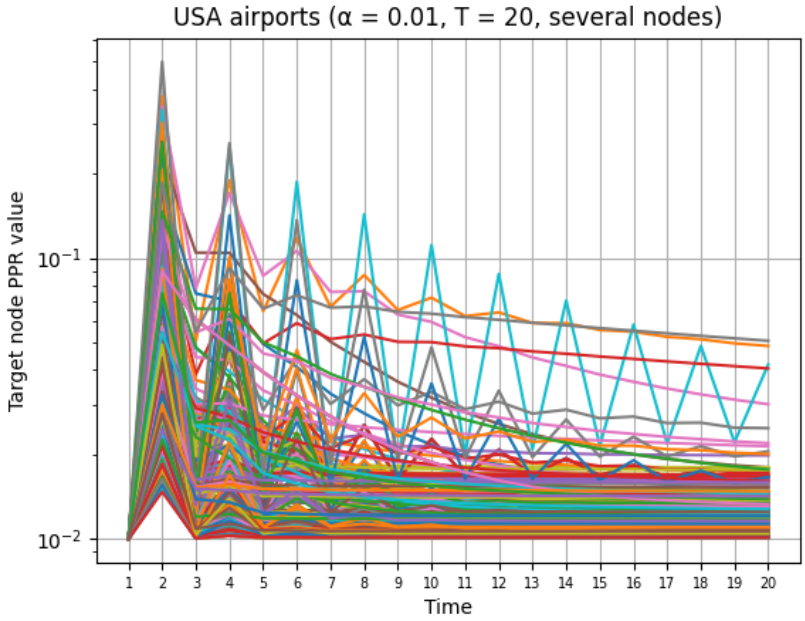


Figure 4.7: Evolution of target node PPR values over time for several nodes of the USA airports network

representations. Based on these analyses, we are able to conclude the ranges of values that are more relevant when it comes to bringing variety to the representations.

Figures 4.8a, 4.9a and 4.10a show the mean target node PPR value over the iterations. Each curve corresponds to a value of α. At each iteration t, the value in each curve is the average of the PPR values of all nodes in the network, where the value of each node at time t is the value the node assumes when it is the target node of PPR. The value of the curve that corresponds to $\alpha = a$ after iteration t is then described by

$$y_a(t) = \frac{1}{|V|} \sum_{u \in V} P_a(u, t) \tag{4.9}$$

where $P_a(u, t)$ is the PPR value of node u after iteration t when u is the target node of PPR with $\alpha = a$.

Two conclusions can be made from Figures 4.8a, 4.9a and 4.10a. The first one is that the higher the value of α, the more the curve flattens, meaning little to no value is contained in the PPR sequence. Instead, the sequence could simply be summarized to be the actual value of α, on average. The second is that the lower the value of α, the more similar the sequences become, i.e. the curves are more closely packed together. This means that the gain in variety in the PPR sequences from using smaller values for α decreases as α decreases.

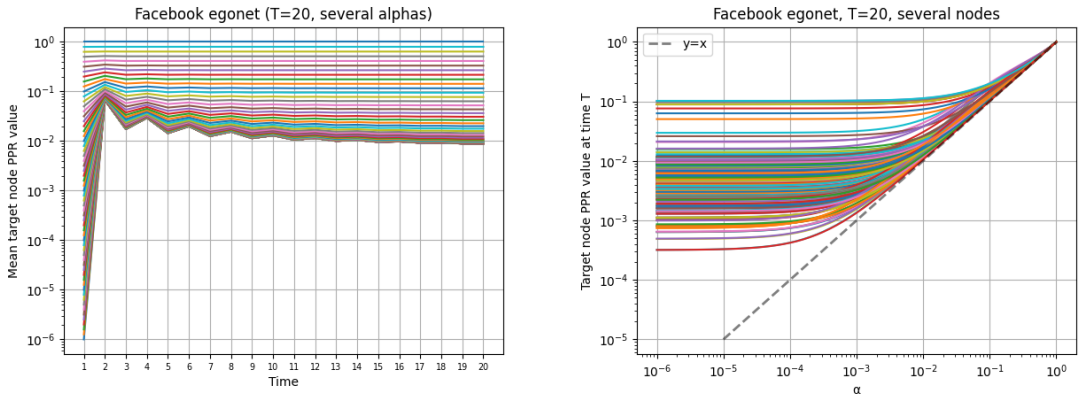
Figures 4.8b, 4.9b and 4.10b also support these conclusions. In all of them, each curve corresponds to a network node, and show the target node PPR value at time T (i.e., the last value in the sequence) for a range of values for α. The value of the curve that corresponds to node u is then described by

$$z_u(a) = P_a(u, T) \tag{4.10}$$

where a is the value of α and $P_a(u, T)$ is the PPR value of node u after iteration T (i.e., the last iteration) when u is the target node of PPR with $\alpha = a$.

In Figures 4.8b, 4.9b and 4.10b, almost all curves become more closely packed together and also simultaneously closer to the line $y = x$ as α approaches 1, meaning the sequences of almost all nodes converge to the value of α, which supports the conclusion that PPR sequences for larger values of α contain less node differentiation. Also, almost all curves become near constant below a certain value of α, which means that the sequences for these different values of α all converge to very similar values, supporting the conclusion that the gain in variety in the PPR sequences from using smaller values for α decreases as α decreases.

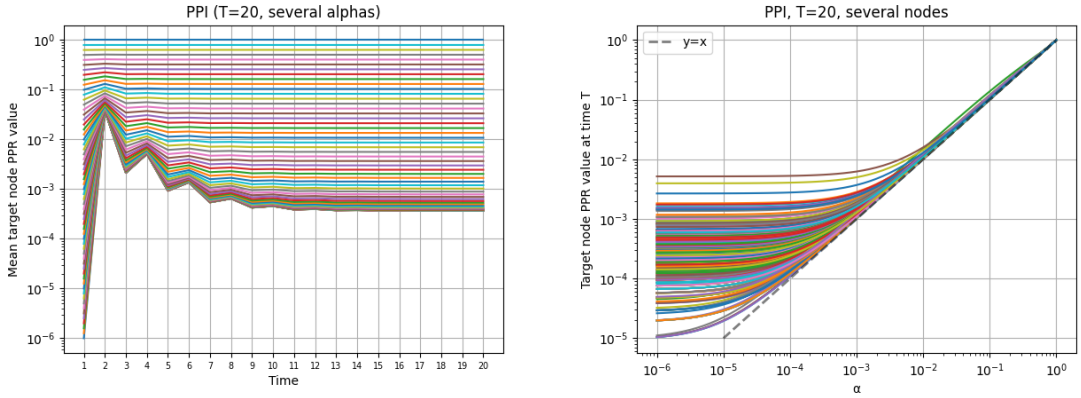
These results are consistent with the behavior of the PPR algorithm. Large values of α make the contributions of the neighborhoods of each node become increasingly irrelevant, since these contributions are weighed by $1-\alpha$. On the opposite



(a) Mean target node PPR value evolution over time for several values of α

(b) Target node PPR value at time T across a range of α values

Figure 4.8: Study of α on PPR values - Facebook egonet



(a) Mean target node PPR value evolution over time for several values of α

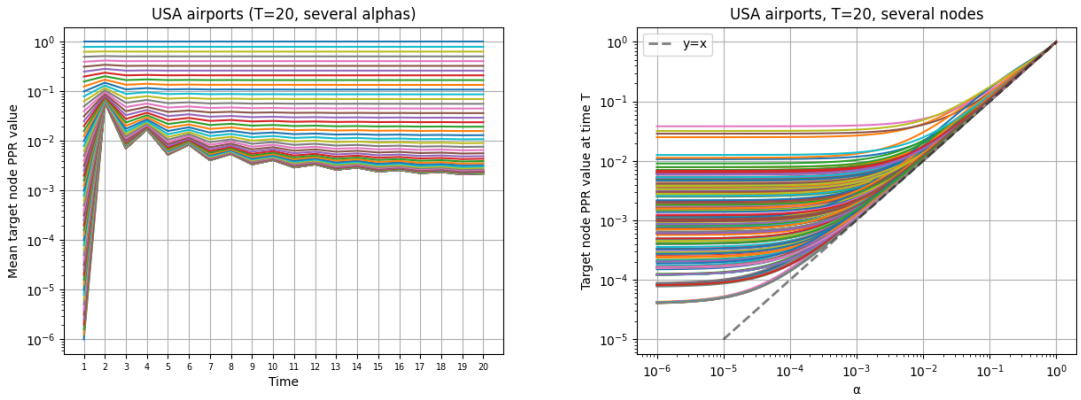
(b) Target node PPR value at time T across a range of α values

Figure 4.9: Study of α on PPR values - PPI

side of the spectrum, very small values of α make the algorithm behave more similarly to the regular PageRank algorithm, with no personalization, which makes the sequences become increasingly similar to one another.

4.4 Robustness to edge removal

Finally, we were interested in studying the robustness of the method to the removal of edges. We adopt the same methodology used in struc2vec [3]. We duplicated the network and linked the two connected components to each other by adding an edge from the node of largest degree to its mirror, as to disturb the structure around each node as little as possible. In particular, we wanted to see how similar the representation for each node would be to the representation for its mirror node, after randomly removing a percentage of the edges of the graph.



(a) Mean target node PPR value evolution over time for several values of α

(b) Target node PPR value at time T across a range of α values

Figure 4.10: Study of α on PPR values - USA airports

We generated PPR representations for all nodes in each mirrored network at different levels of random edge removal, always guaranteeing that the resulting network would still be connected (if the network was not connected. another random set of edges was chosen to be removed). At each one of these levels, we flattened the representations of all nodes so that they would become vectors (instead of matrices) by concatenating all rows sequentially. Then, we computed the Euclidean distance from each node's representation to the representation of their mirror node. We also computed the distance from each node's representation to the representation of a different (randomly chosen) node, so that it would also be possible to compare the distribution of distances between mirror nodes to the overall distribution of distances.

The results can be seen in Figures 4.11 and 4.12, which show the CCDF of these distances, at each level, separately for mirror pairs and random pairs. It is clear that at all levels of edge removal, for both networks, the distances between mirror nodes are consistently smaller than the distances between random pairs of nodes at the same level of removal. Note that the difference in the distances is larger for smaller percentages of edge removal. This means that the method shows a fair amount of robustness to the removal of edges, since even after removing a significant percentage of the edges (e.g., 90%), representations of mirror nodes continue to be closer to each other than to representations of random nodes. In other words, if the method was not at all robust, from the perspective of a node its mirror node should look more like any other node in the graph as the percentage of removed edges grows, eventually becoming indistinguishable, but that is not the case. Also note that the CCDF distributions have a long tail. That is, even though most distances are fairly small, there are still distances that are much greater than the average. This is true both for random pairs and for mirror pairs.

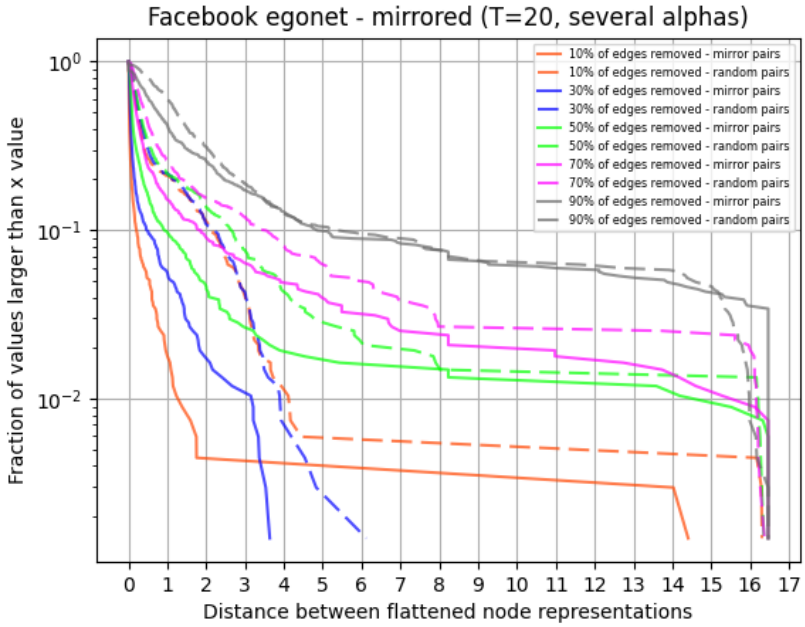


Figure 4.11: CCDFs of distance between flattened mirror node PPR representations (Facebook egonet mirrored)

Unfortunately, we were unable to perform the same experiment with the PPI network, as we could not find a way to keep the network connected, even after removing a small percentage of edges. In order to keep it connected, we would have to abandon a truly random approach, which could distort the results.

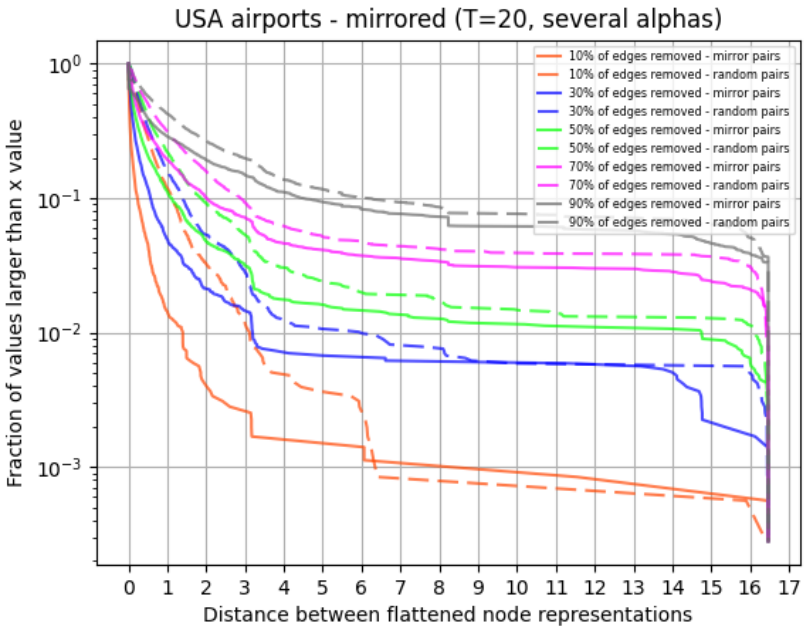


Figure 4.12: CCDFs of distance between flattened mirror node PPR representations (USA airports mirrored)

Chapter 5

Using PPR representations for node classification

A common task that leverages network node representations is node classification. The goal is to use the representations as features to train classifiers, often using a supervised approach. This chapter applies PPR representations to supervised node classification, while also studying the influence of the parameter α and T on the classification accuracy. In addition, we compare our method to well-known prior approaches, more specifically GraphSAGE and struc2vec.

5.1 Datasets and methodology

We chose a classification problem in which the structural role of nodes is key to predicting the correct node labels. struc2vec's GitHub repository [21] contains three graphs for which node labels are also provided. All three of them are airport traffic networks, one already studied in previous Chapters (USA) and two others (Brazil and Europe). Summary metrics for these networks are shown in Table 5.1.

Table 5.1: Summary metrics of airport networks

Metric	Brazil	Europe	USA
Number of nodes	131	399	1190
Number of edges	1074	5995	13599
Minimum degree	1	1	1
Maximum degree	81	202	238
Average degree	16.40	30.05	22.86
Median degree	10	15	6
Average clustering coefficient	0.64	0.54	0.50
Number of connected components (CC)	1	1	3
Size of largest CC	131	399	1186
Diameter of largest CC	4	5	8
Average shortest path length of largest CC	2.19	2.32	3.07

The labels were assigned based on the activity level of each airport, which means

the total number of landings and departures, for Brazil and Europe, and the total number of people that landed on or departed from each airport, for the USA. In all cases, four labels were assigned, based on the quartiles of activity levels, meaning the 25% less active airports were assigned to the first group, and so on, until the 25% most active airports are assigned to the last group. This division leads to a balanced dataset, which is why we use classification accuracy as our only evaluation metric. The reason why these labels are related to the structural role of nodes is because the activity level of an airport does not follow a homophily pattern (i.e., labels of neighbors do not strongly correlate), and is more dependent on whether the airport is located in a large metropolitan area where many people live, in a region that is relevant for some reason (e.g., thriving business scene, common vacation destination), or in a small city. An airport with high levels of activity has flights to several airports with much lower levels, and only a few airports with high levels as well.

Each training experiment throughout this Chapter consists of 20 training executions. Each execution divides the dataset randomly in two parts, one with 80% of the nodes, for training, and the other, with the remaining 20%, for testing. Results are reported for the test set only. We run experiments using different models for learning, in particular Support Vector Classifier (here we use the more general and widely used abbreviation SVM for Support Vector Machines), Logistic Regression, and a Recurrent Neural Network (RNN).

We used linear kernel for SVM. For logistic regression, we used L2 penalty, lbfgs solver, which handles multinomial loss, and 1000 maximum iterations. Finally, RNN uses an extremely simple architecture, consisting of a LSTM layer with 16 units (the dimensionality of its output space), *tanh* activation function, sigmoid recurrent activation function, glorot uniform initializer, orthogonal recurrent initializer, and 0.1 dropout rate, followed by a dense layer with 4 units (same as the number of groups of airports) and softmax activation function, for classification output. The network is trained with a batch size of 128, adam optimizer, and categorical cross-entropy loss, for 100 iterations over the dataset (epochs). Our goal was to focus on the power of the representations, not on the power of the classification methods, thus we have kept their complexities to a minimum, and we did not optimize their various parameters.

Recall that the PPR representations for each node is a matrix, not a vector. So, for SVM and Logistic Regression we concatenate the rows of this matrix, resulting in an uni-dimensional vector, which is what these methods accept as input. Also recall that the number of rows in the representation is the number of values of α that were chosen, and the number of columns is equal to T. For RNN we transpose the second and third axes of the representation matrix, so that each row now corresponds to

an iteration, not a value of α. Since the RNN expects an input in which each row contains the features at a timestep, this transformation allows the RNN to properly use the temporal aspect of the representation.

5.2 Optimizing PPR parameters

First, we studied the effect on the classification accuracies for each airport network as the value for T and the values for α vary. For these experiments, we chose values for T ranging from 10 to 50 in steps of 5, and seven different values for the exponent interval of α, whose exponent is always between -4 and -1, with base 10. In other words, if the value for the exponent interval is 0.5, α will assume a total of seven values: 10^{-4}, $10^{-3.5}$, 10^{-3}, $10^{-2.5}$, 10^{-2}, $10^{-1.5}$, and 10^{-1}. The chosen values for the exponent interval are 0.01, 0.025, 0.05, 0.1, 0.2, 0.5 and 1.

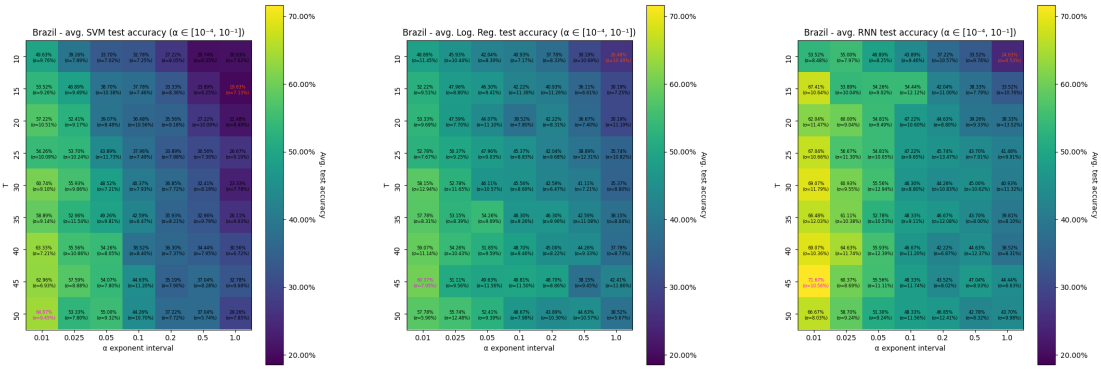
The values for T are long enough for the sequences to get closer to convergence values, but not so long that the difference in values becomes irrelevant. From Figures 4.8a, 4.9a and 4.10a we conclude that $T = 10$ would be a good value for the minimum number of iterations required as, on average for those three networks, sequences are near convergence after the 10th iteration. We chose $T = 50$ as an upper bound not to explode the size of the representations.

We chose the range $[10^{-4}, 10^{-1}]$ for α based on Figures 4.8b, 4.9b and 4.10b. Below 10^{-4}, sequences tend to converge to the same value as they do for higher exponents. Above 10^{-1}, sequences are too much alike and close to the $y = x$ line. As for the values of $\alpha 4$ exponent interval, our goal was to have enough variety in the number of values for α. An exponent interval of 1 leads to 4 different values for α, while an exponent interval of 0.01 leads to 401 different values.

Each experiment was run with a value for T and a value for the α exponent interval. Figures 5.1, 5.2 and 5.3 show the results for each combination, for each classification method on each airport network. In each figure, the row corresponds to a value for T, increasing as we go down the rows, and each column corresponds to a value for the α exponent interval, increasing from left to right. The color of each cell belongs to a scale that starts just below 20% (violet) and ends just above 70% (yellow-ish green). The value of each cell corresponds to the average accuracy of the 20 independent executions in the corresponding experiment.

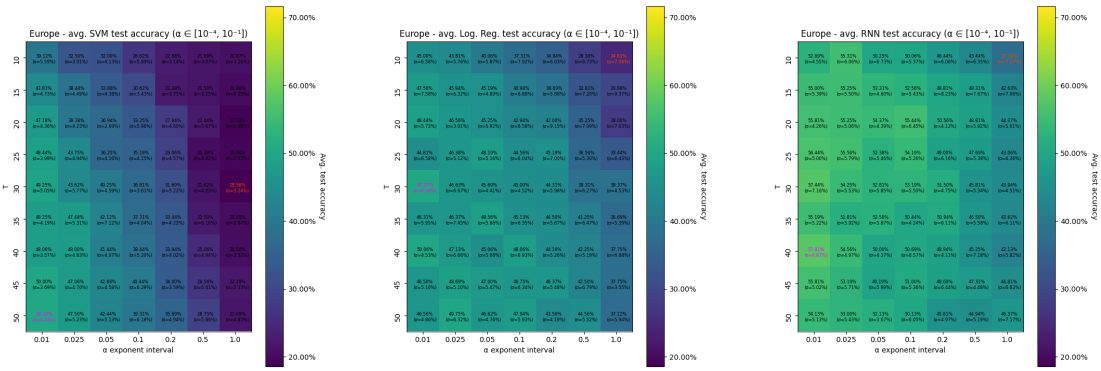
Even though these figures are too small, effectively making it impossible to read individual values in a cell, the most important information is the trend of the results, represented by the colors of the cells, which is still discernible. These figures, in a larger size, and several others are available in the appendix, for further inspection.

It is evident from these figures that lowering the value for the α exponent interval (i.e., increasing the number of values for α) consistently leads to better results, for



(a) SVM (b) Logistic Regression (c) RNN

Figure 5.1: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - Brazil airports

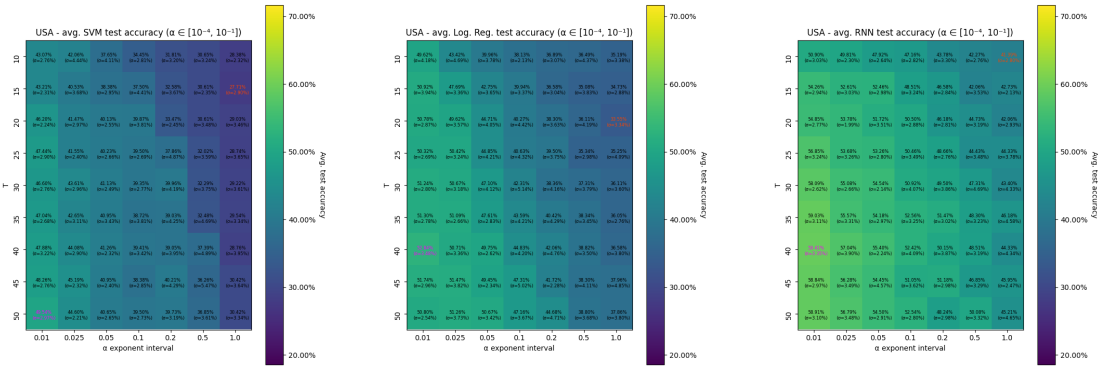


(a) SVM (b) Logistic Regression (c) RNN

Figure 5.2: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - Europe airports

all three networks. In other words, the colors of the cells on the left are always closer to the top of the scale than the colors of the cells on the right are. It is also clear, while not as strong, that increasing the value of T, for a fixed value of the α exponent interval (i.e., in a column), usually leads to better results, for all three networks.

These figures also allow us to conclude that RNN showed the best classification accuracy, while SVM was the worst. This is most likely explained by the fact that RNN has the capacity to utilize the temporal aspect of the representations, while SVM and logistic regression treat all values in the sequences as temporally independent variables.



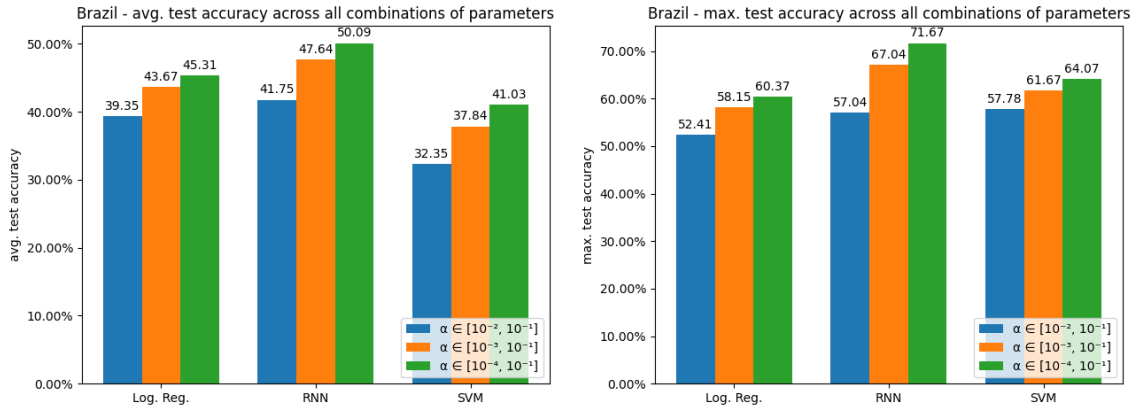
(a) SVM (b) Logistic Regression (c) RNN

Figure 5.3: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - USA airports

5.3 Effect of the range of exponents for α

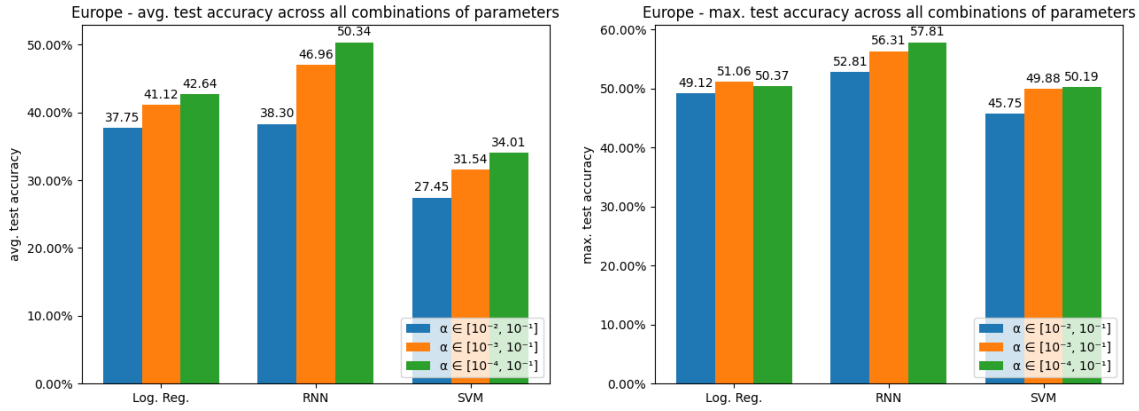
Our second study is on the range of values from which α values can be chosen. In the previous Section we used the $[10^{-4}, 10^{-1}]$ range. However, it could be the case that using shorter ranges would not lead to significant losses in performance, if they hold enough relevant information for differentiating the nodes. Note that this will shorten the representation, since it depends on the number of α values. For this reason, we run the same experiments using the ranges $[10^{-3}, 10^{-1}]$ and $[10^{-2}, 10^{-1}]$.

Figures 5.4, 5.5 and 5.6 compare the results obtained when using these three ranges, for each airport network, with each classification method. The values of the bars refer to all combinations of T and α exponent interval for that scenario. It is clear that, in almost all cases, allowing a wider range leads to better results, both for the average performance and for the maximum performance. The only exception is maximum Logistic Regression accuracy on the Europe airports network (Figure 5.5b), where the $[10^{-3}, 10^{-1}]$ range achieves the best result. However, the difference between the shorter range and the intermediate range is usually larger than that between the intermediate range and the wider one, meaning the $[10^{-3}, 10^{-1}]$ range still holds much of the variety found in the $[10^{-4}, 10^{-1}]$ range, but a lot more is lost when only the $[10^{-2}, 10^{-1}]$ range is considered. This finding is in accordance with the results shown in Figures 4.8b, 4.9b and 4.10b, since they show that there is less node differentiation (i.e., the curves are closer to each other) in the $[10^{-2}, 10^{-1}]$ range than there is in the $[10^{-3}, 10^{-2}]$ and the $[10^{-4}, 10^{-3}]$ ranges. Also, once again RNN achieves the best performance in almost all of the scenarios, and SVM the worst.



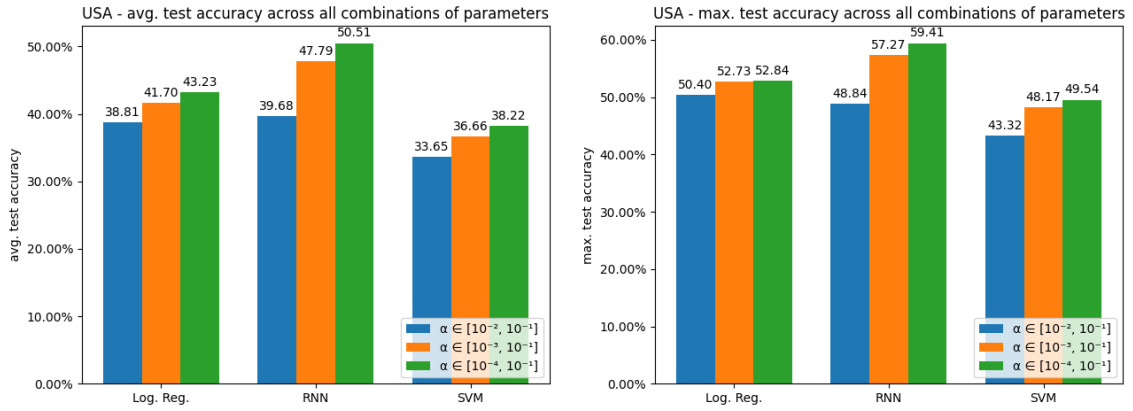
(a) Average (b) Maximum

Figure 5.4: Test accuracy of all classification methods across all combinations of parameters (Brazil airports)



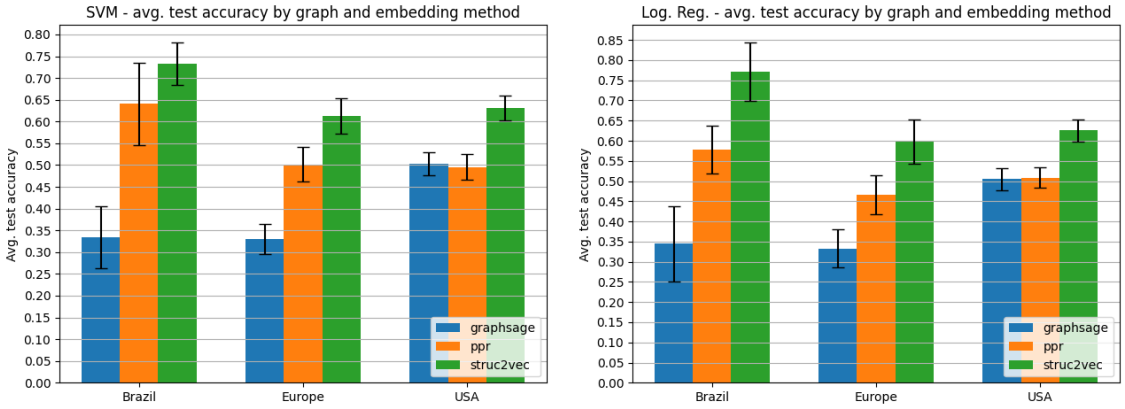
(a) Average (b) Maximum

Figure 5.5: Test accuracy of all classification methods across all combinations of parameters (Europe airports)



(a) Average (b) Maximum

Figure 5.6: Test accuracy of all classification methods across all combinations of parameters (USA airports)



(a) SVM

(b) Logistic Regression

Figure 5.7: Average classification test accuracy by graph and embedding method

5.4 Comparing embedding methods

Finally, we compare the classification accuracies of GraphSAGE, struc2vec and PPR. For PPR, we use the $[10^{-4}, 10^{-1}]$ range, as it showed the best results, with α exponent interval set to 0.01 and T set to 50, in all cases, for consistency. GraphSAGE representations were generated in unsupervised mode with a dimension of 128, using mean-based aggregator, 1000 maximum total steps and validation every 10 steps. struc2vec representations were generated with a dimension of 128, walk length of 80, number of walks set to 10, window size 10, for 5 epochs, using all three available optimizations, limiting to 5 layers (for optimization number 3).

Figure 5.7 shows the comparison when using SVM and logistic regression as classification methods. RNN was excluded, since the representations generated by GraphSAGE and struc2vec do not have a temporal or sequential aspect to them, and it would not make sense to treat them as so. Note that struc2vec clearly outperforms the other methods with both classifiers, for all networks. PPR representations perform significantly better than GraphSAGE, except for the USA airports network, for which PPR performs the same or slightly worse. This result indicates that PPR representations are better suited for classifiers that can handle features that have a temporal dependence. Clearly, PPR representations have a strong temporal component which can be explored by an adequate classifier, such as an RNN.

Figure 5.8 shows the best average test accuracy achieved by any of the three classifiers for each embedding method on all airport networks. For PPR, we use the $[10^{-4}, 10^{-1}]$ range, with α exponent interval set to 0.01 and T set to 50, in all cases, for consistency.

Logistic regression shows the best results for GraphSAGE in all cases, as does RNN for PPR representations. For struc2vec, Logistic Regression shows the best result on the Brazil airports network, but SVM shows the best results for Europe

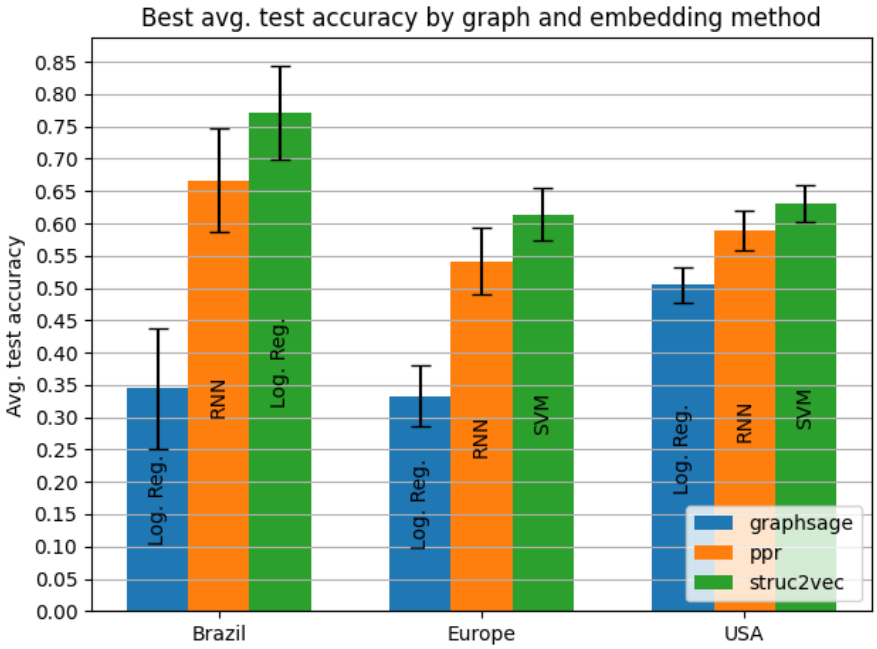


Figure 5.8: Best average test accuracy by graph and embedding method (PPR: $\alpha \in [10^{-4}, 10^{-1}]$, α exponent interval 0.01, $T = 50$)

and USA. Note that while struc2vec shows superior performance for all networks, PPR is a relatively close second, considering the relative large margins of error for the smaller networks. In any case, the representation generated by struc2vec seem more adequate for learning the labels of these network nodes.

Moreover, when compared to Figure 5.7, it is clear from Figure 5.8 that PPR representations benefit considerably from a classifier that is capable of handling features with a strong temporal aspect. With RNN, PPR average test accuracy had a 4% increase for the Brazil airports network when compared to the previous best result, achieved with SVM (Figure 5.7a). For the Europe airports network, there was a 8% increase over SVM (Figure 5.7a), and for the USA airports network there was a 16% increase over Logistic Regression (Figure 5.7b).

Chapter 6

Conclusion and Future Work

Several methods for generating vector representations for nodes in graphs have been developed in recent years, each with their own particularities, strengths and limitations. Some are more focused on capturing network proximity and homophily, while others are more focused on capturing the local structure around the node, regardless of their labels or labels of other nodes in the network. A common characteristic is that almost all of these methods rely on randomness in order to generate the node representations, and thus generate random representations.

This work has characterized two of these methods (GraphSAGE and struc2vec) on three different networks in order to assess the consistency of the representations generated by them from one execution to another. Several different metrics, as well as their robustness to the removal of nodes and edges have been evaluated. We have shown that there is a non-negligible level of variation across executions, meaning the representation for a node varies, which is a direct consequence of the randomness that is inherent to these methods.

The main contribution of this work is a method for generating representations for nodes based on sequences obtained from the iterative process of computing Personalized PageRank. The PPR algorithm generates sequences that depend strictly on the structure of the network and each automorphism group has an unique sequence, acting as a "signature" for the structural identity of network nodes, that is, the role or function each node plays in the network. This means that two nodes with the exact same structural role (i.e., belong to the same automorphism group) will always have identical representations, unlike most prior methods, which can generate representations that are similar, but not identical.

We have also shown that the proposed method is capable of differentiating the many structural identities of nodes in a network. The PPR representations are deterministic and do not require any random factors, which means it is consistent across executions for the same input parameters. The method has also been evaluated to assess its robustness to edge removals, and a fair amount of robustness has

been shown.

The proposed framework is parsimonious (two parameters) and flexible. It allows one to choose the set of nodes for which representations will be generated, and even a single node, unlike the other methods which require representations to be generated for all nodes (or a subset of them), as their representations are directly dependent on the context created from the entire network. The framework also allows to control the size of the resulting representations through the set of values for the return rate α and the value for the horizon T. The effect of these parameters on the generated sequences was also studied, indicating the range of parameters that are most suited for generating better representations.

We have studied the performance of PPR representations on a classification task, for several combinations of parameters, on three airport traffic networks, using three different classification algorithms. Our conclusion is that using a larger set of values of α across a wider range leads to the best results. However, this leads to representations whose sizes are quite large, so a compromise is necessary. It was also concluded that, even though not as strongly, higher values for the horizon T lead to better results. Considering the three investigated classification methods, across all combinations of parameters, RNN proved to be the best, much due to its ability to utilize the temporal aspect of the PPR representations. When compared to struc2vec on classification accuracy, the proposed method did not achieve better results, even with representations of considerably larger sizes, but it did achieve better results than GraphSAGE, which indicates its ability to capture the structural identity of nodes well enough.

The proposed methodology is innovative, and quite different from those of struc2vec and GraphSAGE. Also, the method uses the entire sequence of values assumed by the target node in the iterative procedure to compute the PPR, which is very unusual, as usually the focus is on the final ranks of nodes.

6.1 Future Work

Different studies could be conducted in order to better assess and enhance the proposed methodology. In what follows we outline a few directions for future work.

- Explore different ranges for the values of α. In this work, we have mainly explored the $[10^{-4}, 10^{-1}]$ range, but there can still be value from using values lower than 10^{-4} or higher than 10^{-1} for α. Our study indicated that the final value of the PPR sequences stagnates for values of α below a certain threshold, but the same might not be true for the intermediate values of the PPR sequences.

- Explore the effects of using narrower ranges for the values of α. Instead of keeping the higher orders of magnitude, it would be interesting to study the effects of keeping the lower ones, where there is more node differentiation. For example, to assess if there is still a decline in classification accuracy when using the $[10^{-4}, 10^{-3}]$ range when compared to the $[10^{-4}, 10^{-1}]$ range, as happened when using the $[10^{-2}, 10^{-1}]$ range.

- Explore other approaches to determine the values for α, in particular approaches that are node dependent or network dependent. For example, an approach that detects when PPR sequences are becoming too similar for a node (e.g., using $\alpha = 10^{-5}$ and $\alpha = 10^{-5.1}$ yields sequences that are essentially the same, so lower values are not taken into account, or the α exponent interval is increased dynamically). Another possibility is an approach that makes a decision regarding values for α based on metrics of the network, such as number of nodes and edges, mean degree, or clustering.

- Devise an optimized version of PPR that would allow the method to be run on much larger networks. This could leverage efficient distributed algorithms for computing PPR, which is an active area of research.

- Apply the method to a different classification task, considering different networks and node labels, as well as different problems, such as graph matching or network alignment. In particular, PPR representations could succeed in the graph matching problem given its strong dependency on local structure and relative robustness to edge removal.

References

[1] GROVER, A., LESKOVEC, J. "Node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, p. 855–864, New York, NY, USA, 2016. Association for Computing Machinery. ISBN: 9781450342322. doi: 10.1145/2939672.2939754. Disponível em: <https://doi.org/10.1145/2939672.2939754>.

[2] HAMILTON, W., YING, Z., LESKOVEC, J. "Inductive Representation Learning on Large Graphs". In: Guyon, I., Luxburg, U. V., Bengio, S., et al. (Eds.), *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pp. 1024–1034, 2017. Disponível em: <http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf>.

[3] RIBEIRO, L. F., SAVERESE, P. H., FIGUEIREDO, D. R. "*Struc2vec*: Learning Node Representations from Structural Identity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, p. 385–394, New York, NY, USA, 2017. Association for Computing Machinery. ISBN: 9781450348874. doi: 10.1145/3097983.3098061. Disponível em: <https://doi.org/10.1145/3097983.3098061>.

[4] PAGE, L., BRIN, S., MOTWANI, R., et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66, Stanford InfoLab, November 1999. Disponível em: <http://ilpubs.stanford.edu:8090/422/>. Previous number = SIDL-WP-1999-0120.

[5] MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space". 2013.

[6] MIKOLOV, T., SUTSKEVER, I., CHEN, K., et al. "Distributed Representations of Words and Phrases and their Compositionality". In: Burges, C. J. C., Bottou, L., Welling, M., et al. (Eds.), *Advances in Neural*

Information Processing Systems 26, Curran Associates, Inc., pp. 3111–3119, 2013. Disponível em: <http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>.

[7] BENGIO, Y., DUCHARME, R., VINCENT, P., et al. "A neural probabilistic language model", *Journal of machine learning research*, v. 3, n. Feb, pp. 1137–1155, 2003.

[8] PEROZZI, B., AL-RFOU, R., SKIENA, S. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, p. 701–710, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450329569. doi: 10.1145/2623330.2623732. Disponível em: <https://doi.org/10.1145/2623330.2623732>.

[9] WANG, D., CUI, P., ZHU, W. "Structural Deep Network Embedding". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, p. 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery. ISBN: 9781450342322. doi: 10.1145/2939672.2939753. Disponível em: <https://doi.org/10.1145/2939672.2939753>.

[10] DONNAT, C., ZITNIK, M., HALLAC, D., et al. "Learning Structural Node Embeddings via Diffusion Wavelets". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, p. 1320–1329, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450355520. doi: 10.1145/3219819.3220025. Disponível em: <https://doi.org/10.1145/3219819.3220025>.

[11] YING, R., HE, R., CHEN, K., et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems", *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul 2018. doi: 10.1145/3219819.3219890. Disponível em: <http://dx.doi.org/10.1145/3219819.3219890>.

[12] CHEN, J., MA, T., XIAO, C. "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling". 2018.

[13] DAI, H., KOZAREVA, Z., DAI, B., et al. "Learning Steady-States of Iterative Algorithms over Graphs". v. 80, *Proceedings of Machine Learning*

Research, pp. 1106–1114, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. Disponível em: <http://proceedings.mlr.press/v80/dai18a.html>.

[14] HAMILTON, W. L., YING, R., LESKOVEC, J. "Representation Learning on Graphs: Methods and Applications". 2018.

[15] CAI, H., ZHENG, V. W., CHANG, K. C. "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications", *IEEE Transactions on Knowledge and Data Engineering*, v. 30, n. 9, pp. 1616–1637, 2018.

[16] WU, Z., PAN, S., CHEN, F., et al. "A Comprehensive Survey on Graph Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[17] TANG, J., QU, M., WANG, M., et al. "LINE: Large-Scale Information Network Embedding". In: *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, p. 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee. ISBN: 9781450334693. doi: 10.1145/2736277.2741093. Disponível em: <https://doi.org/10.1145/2736277.2741093>.

[18] KIPF, T. N., WELLING, M. "Semi-Supervised Classification with Graph Convolutional Networks". 2017.

[19] SHCHUR, O., MUMME, M., BOJCHEVSKI, A., et al. "Pitfalls of Graph Neural Network Evaluation", *CoRR*, v. abs/1811.05868, 2018. Disponível em: <http://arxiv.org/abs/1811.05868>.

[20] "Example data on GraphSAGE's public github page". https://github.com/williamleif/GraphSAGE/tree/master/example_data.

[21] "Network datasets on struc2vec's public github page". https://github.com/leoribeiro/struc2vec/tree/master/graph.

[22] "Stanford Large Network Dataset Collection - Social circles: Facebook". https://snap.stanford.edu/data/ego-Facebook.html.

[23] MYERS, J. L., WELL, A., LORCH, R. F. *Research design and statistical analysis*. Routledge, 2010.

Appendix A

Other figures

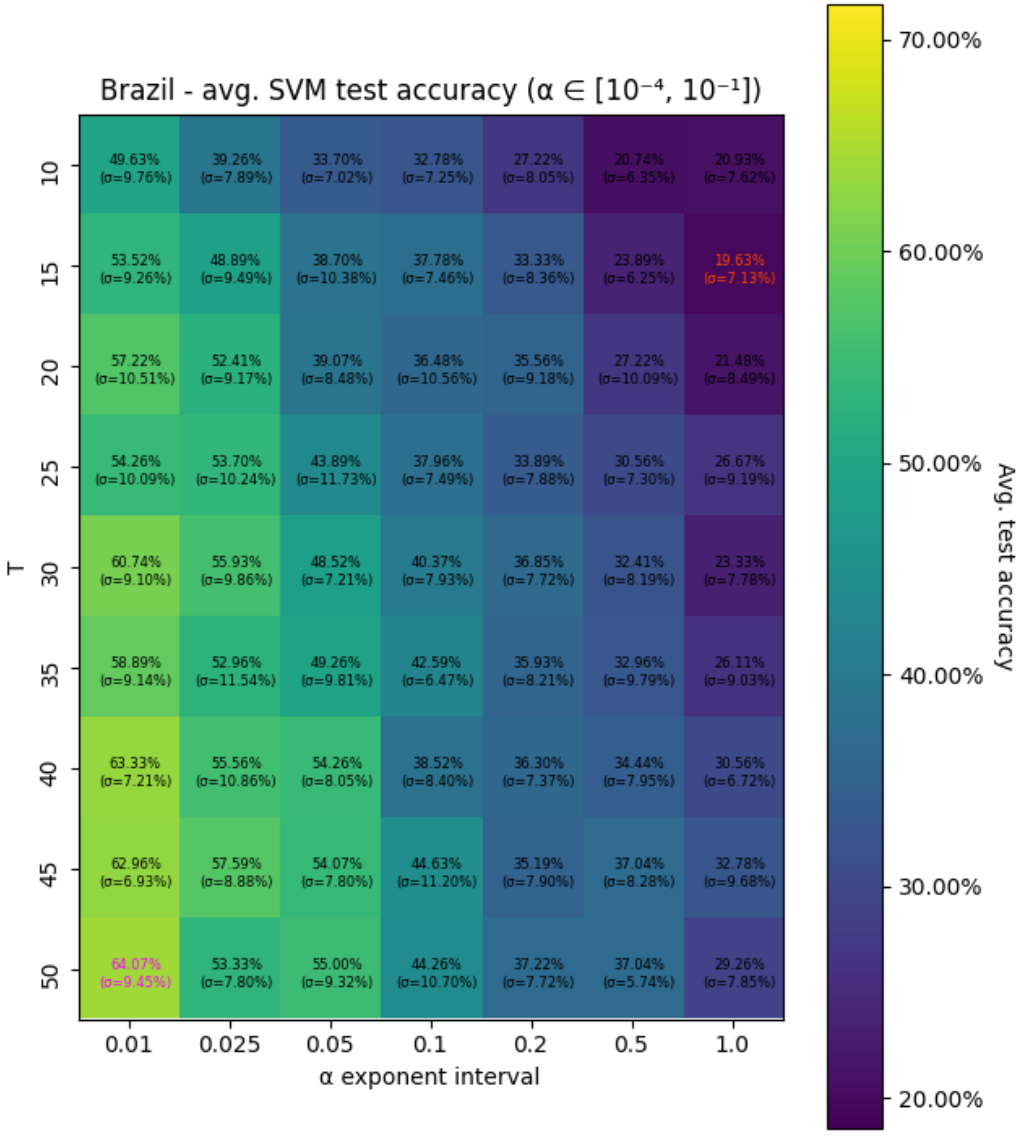


Figure A.1: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - SVM - Brazil airports

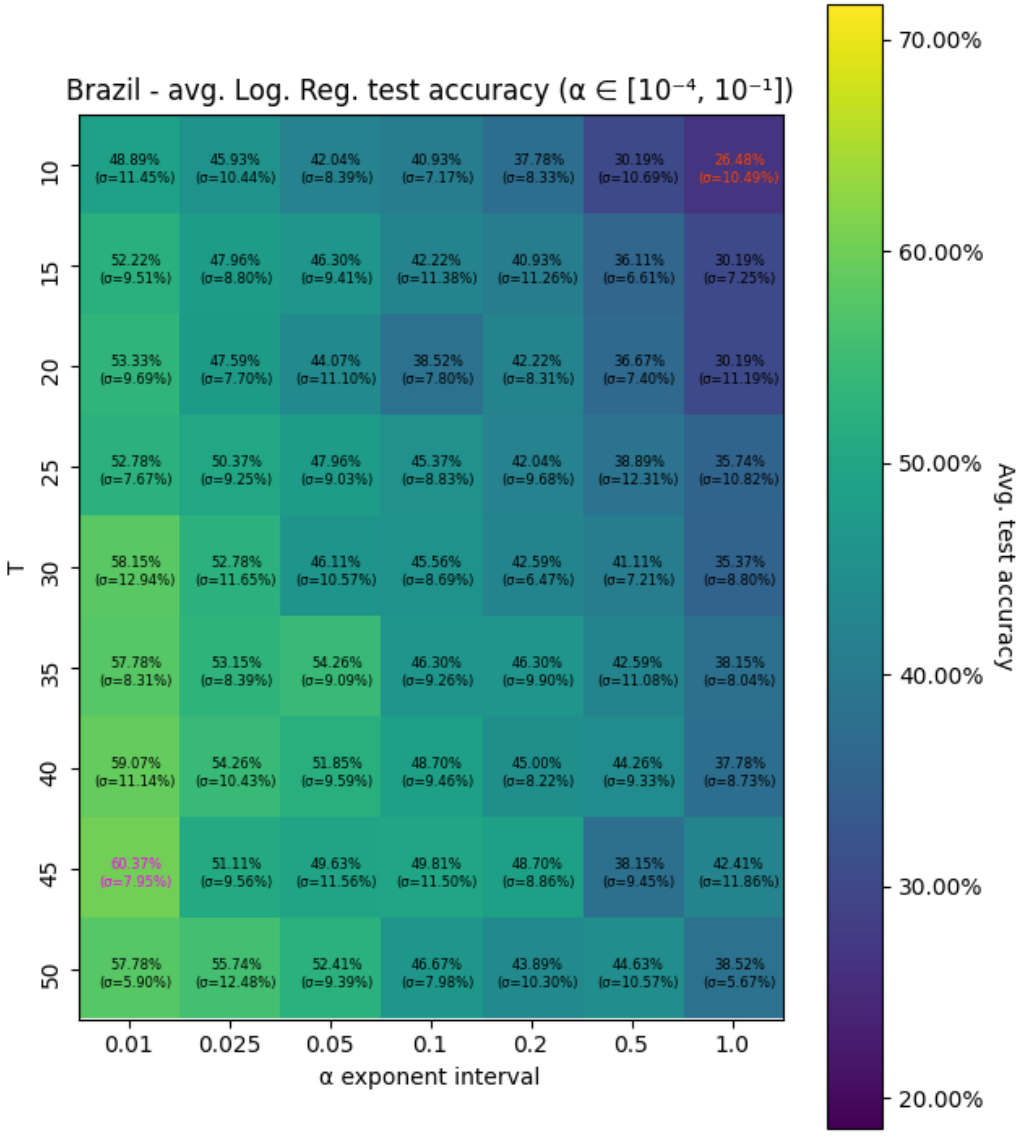


Figure A.2: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - Logistic Regression - Brazil airports

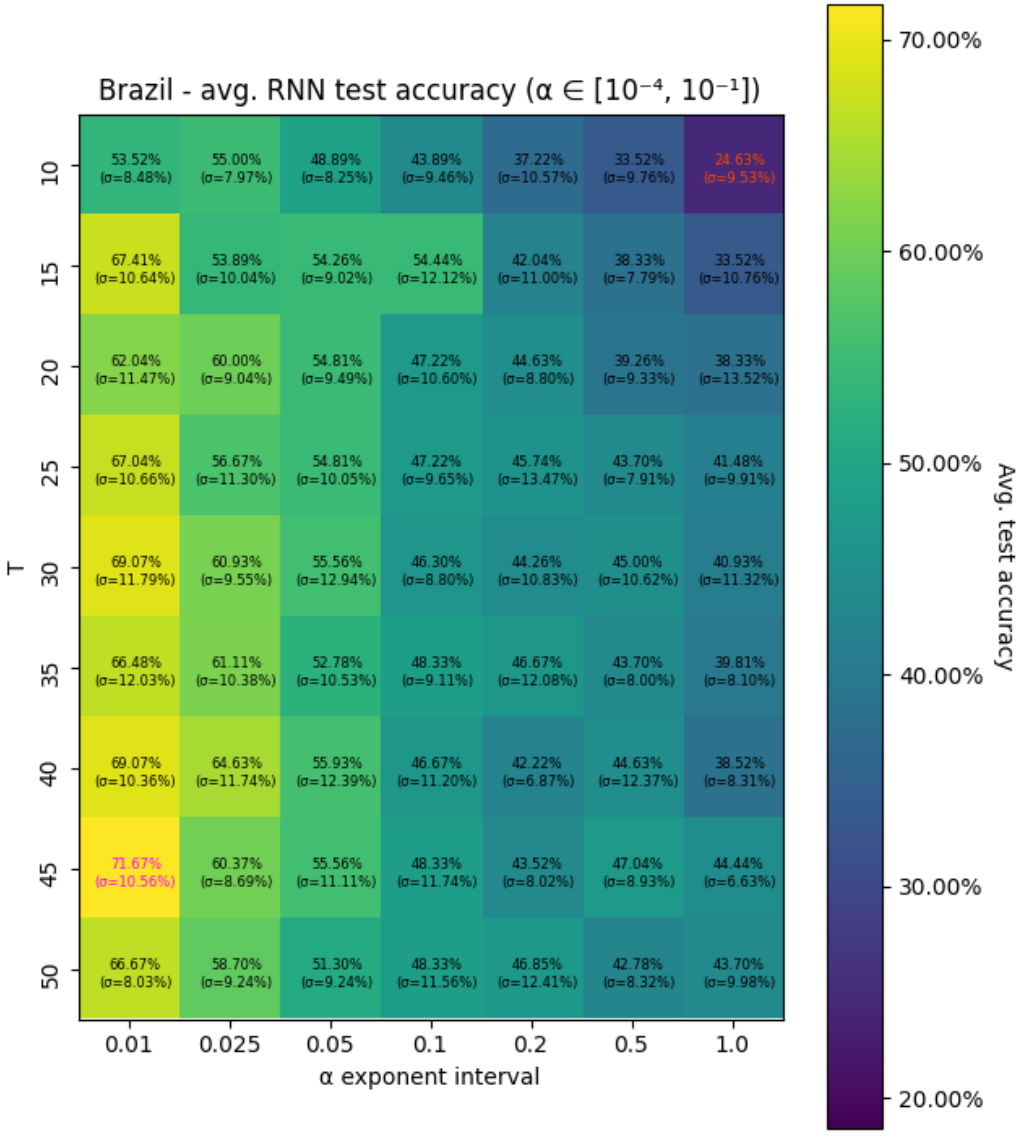


Figure A.3: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - RNN - Brazil airports

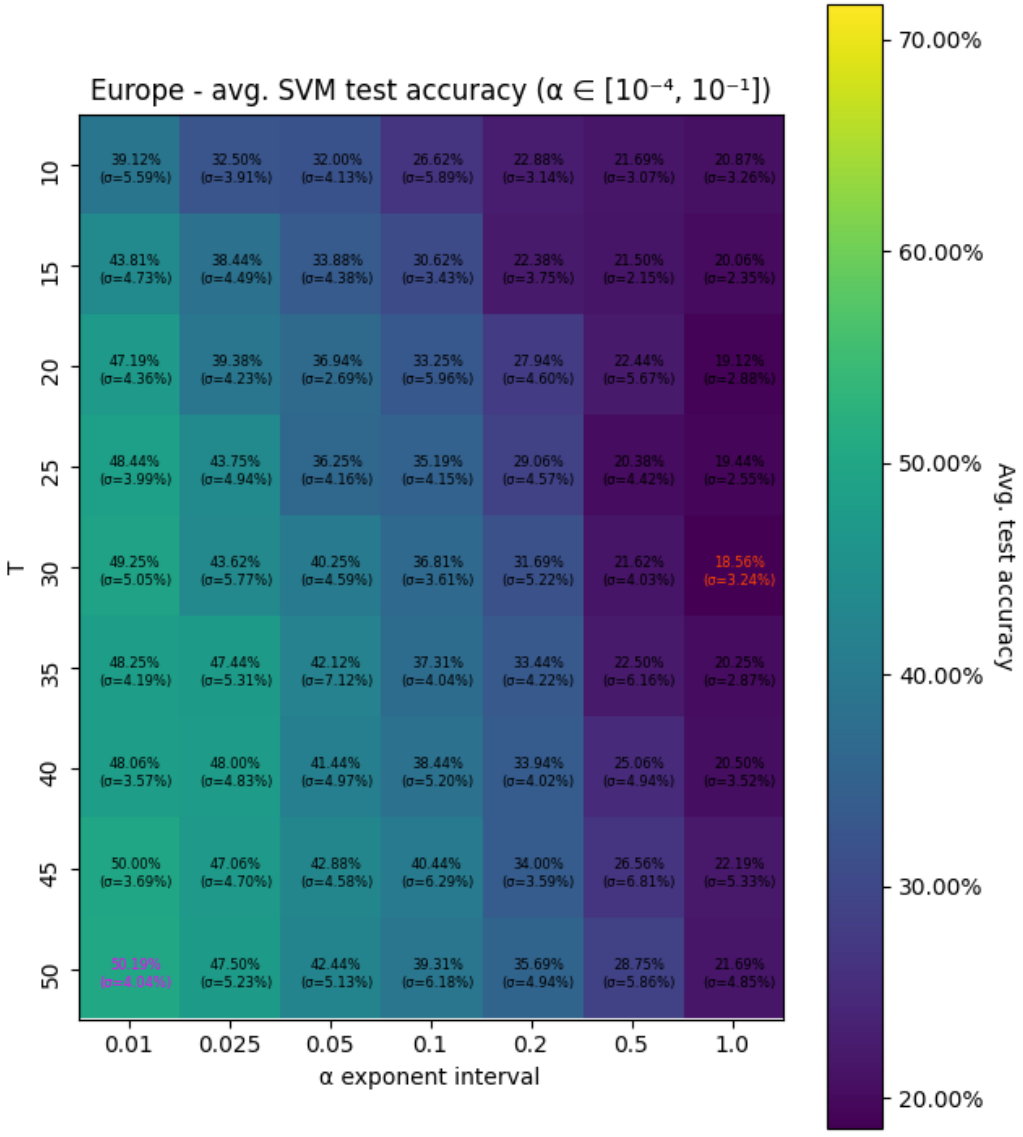


Figure A.4: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - SVM - Europe airports

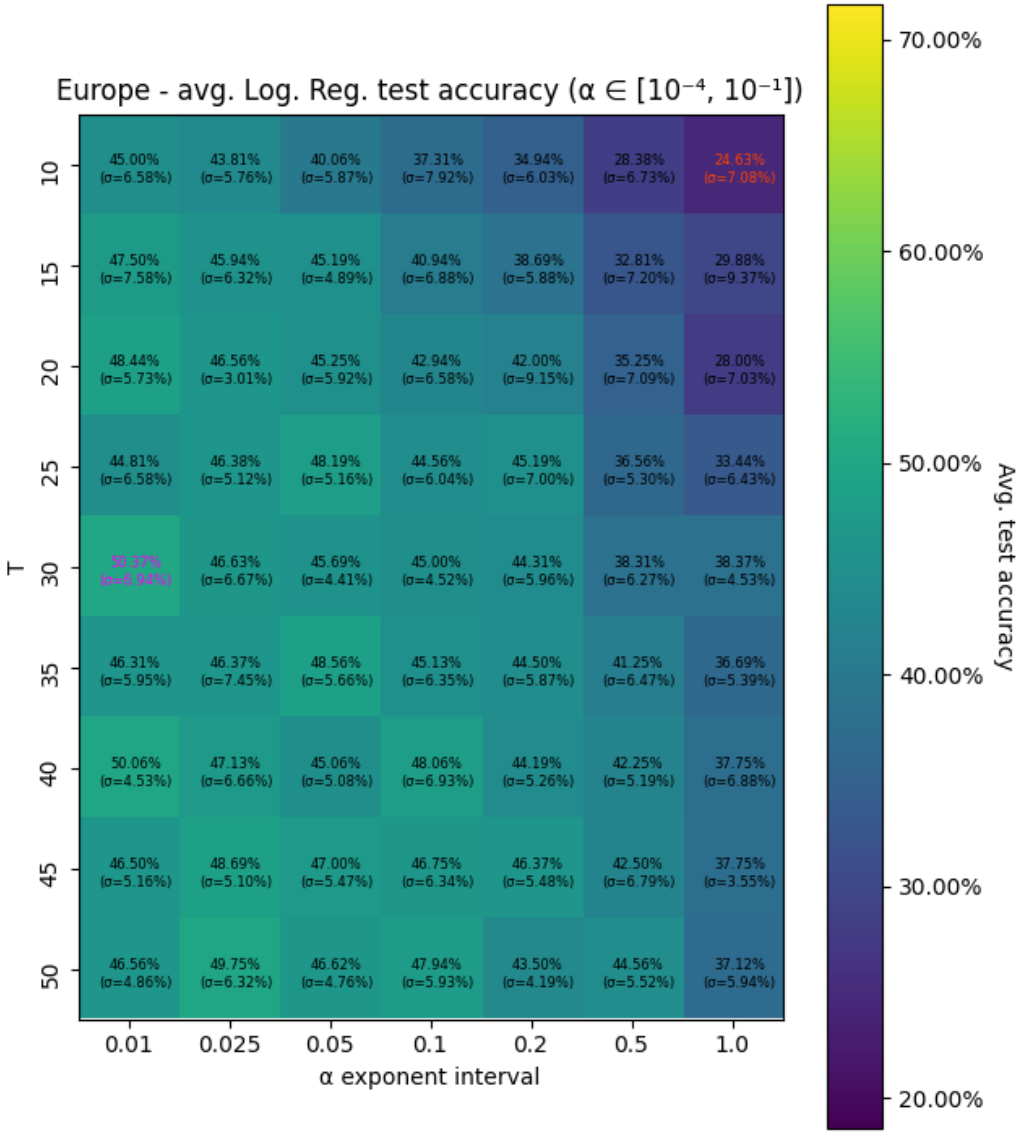


Figure A.5: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - Logistic Regression - Europe airports

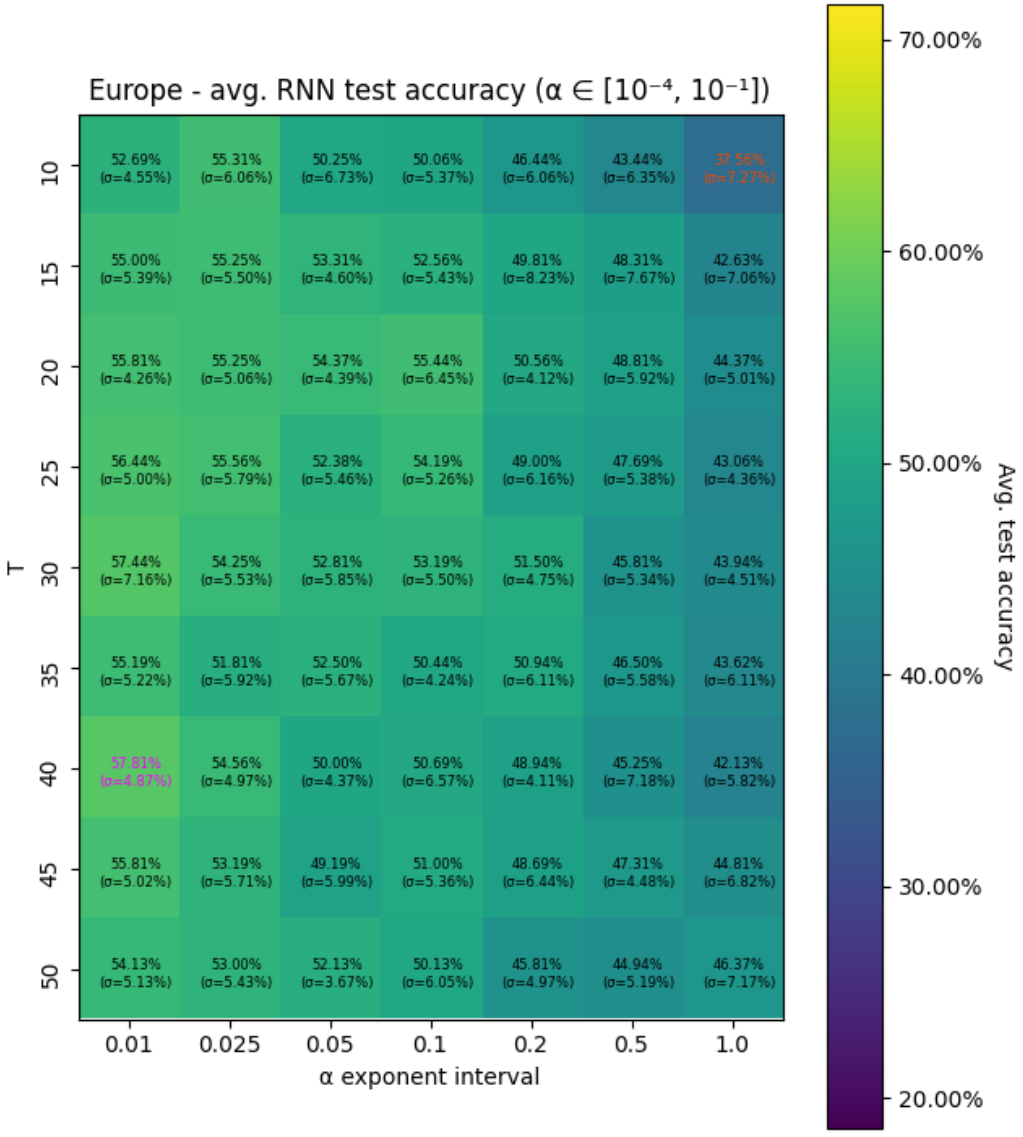


Figure A.6: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - RNN - Europe airports

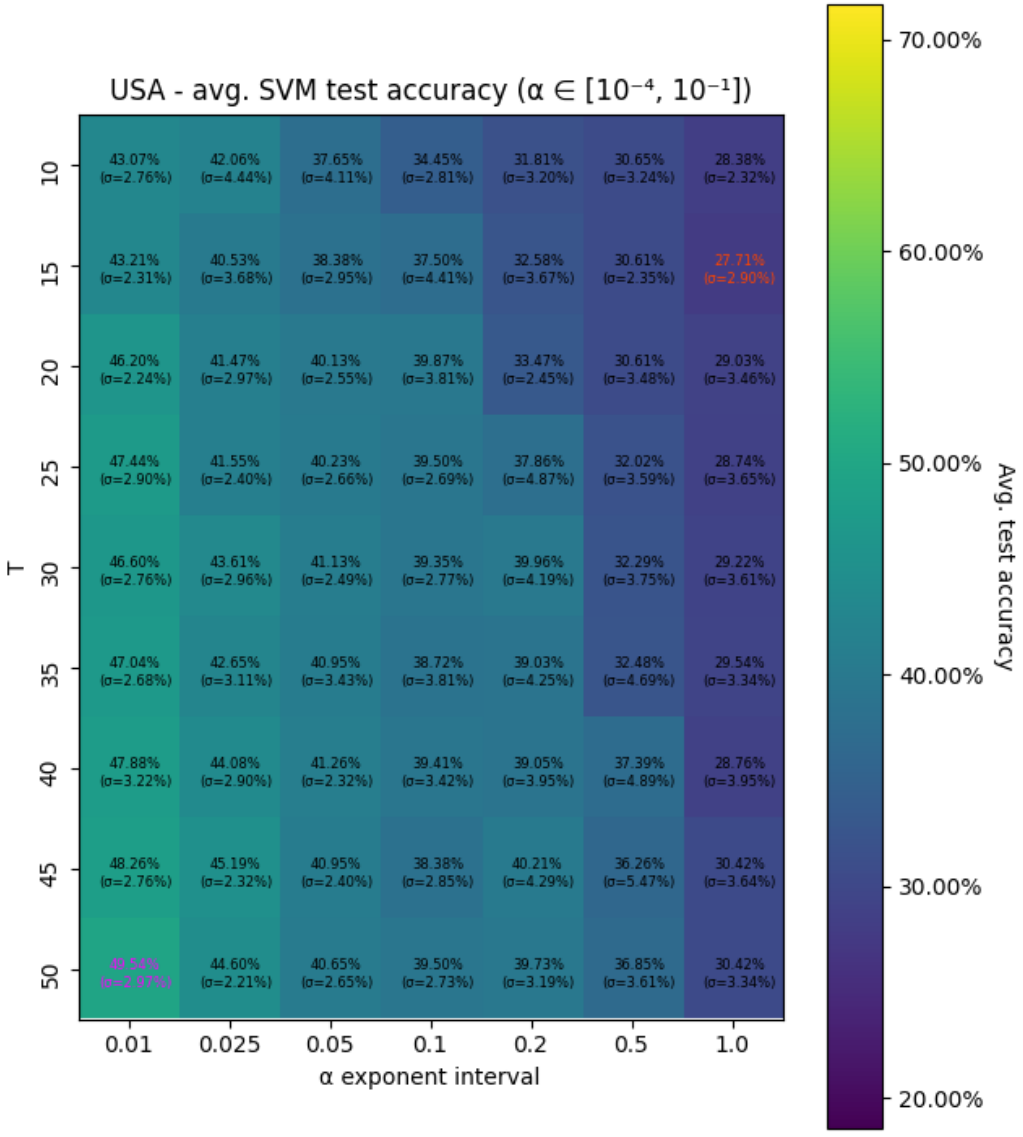


Figure A.7: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - SVM - USA airports

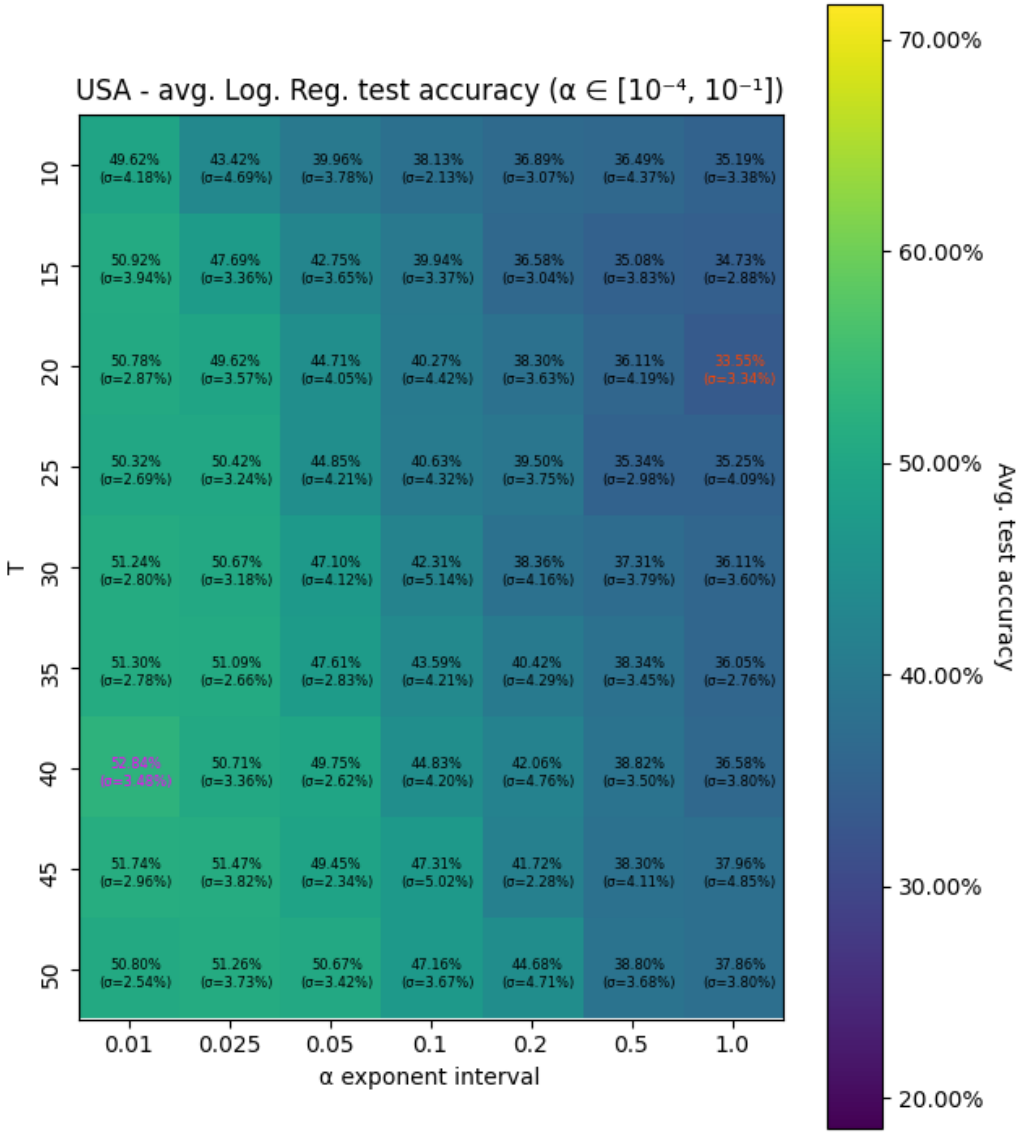


Figure A.8: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - Logistic Regression - USA airports

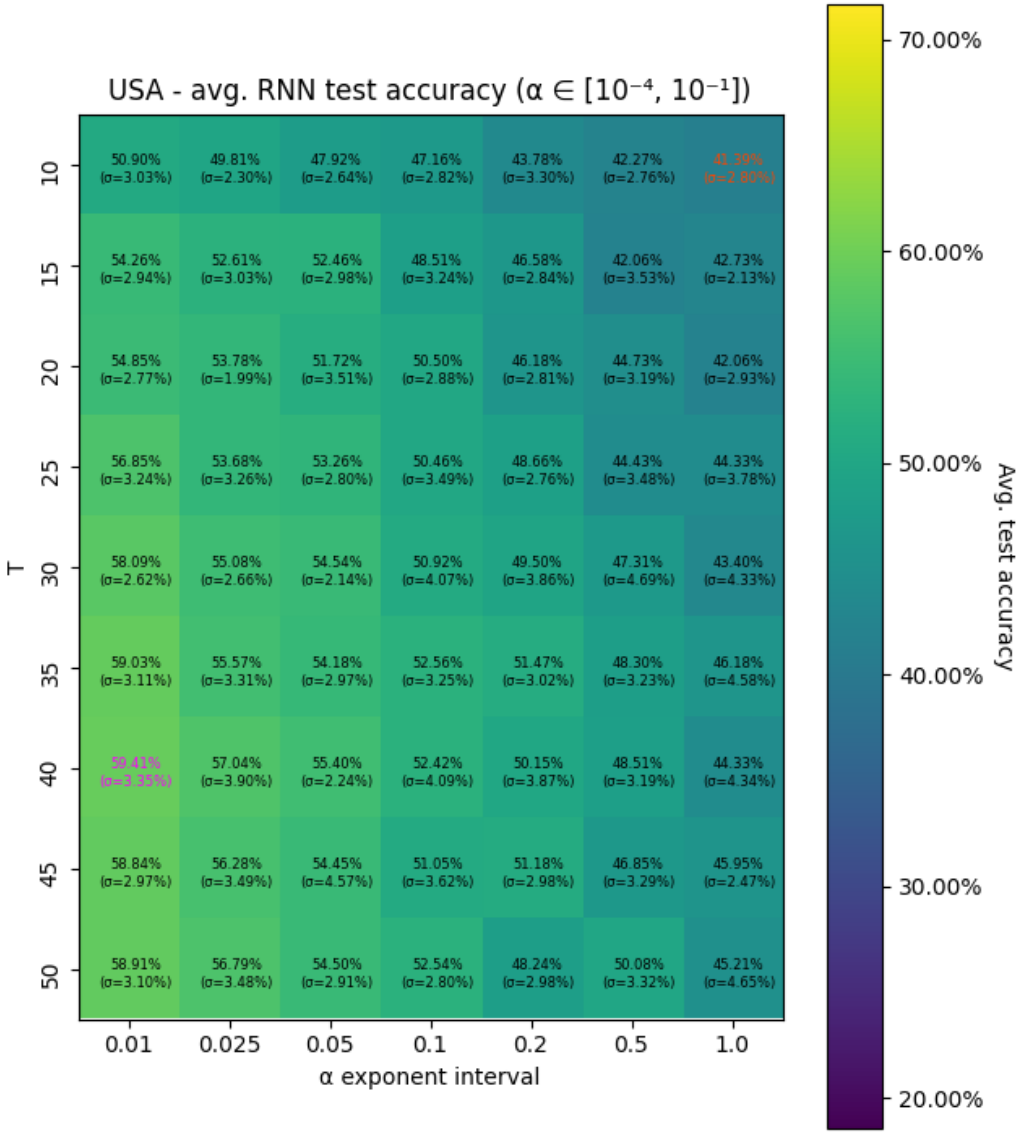


Figure A.9: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-4}, 10^{-1}]$) - RNN - USA airports

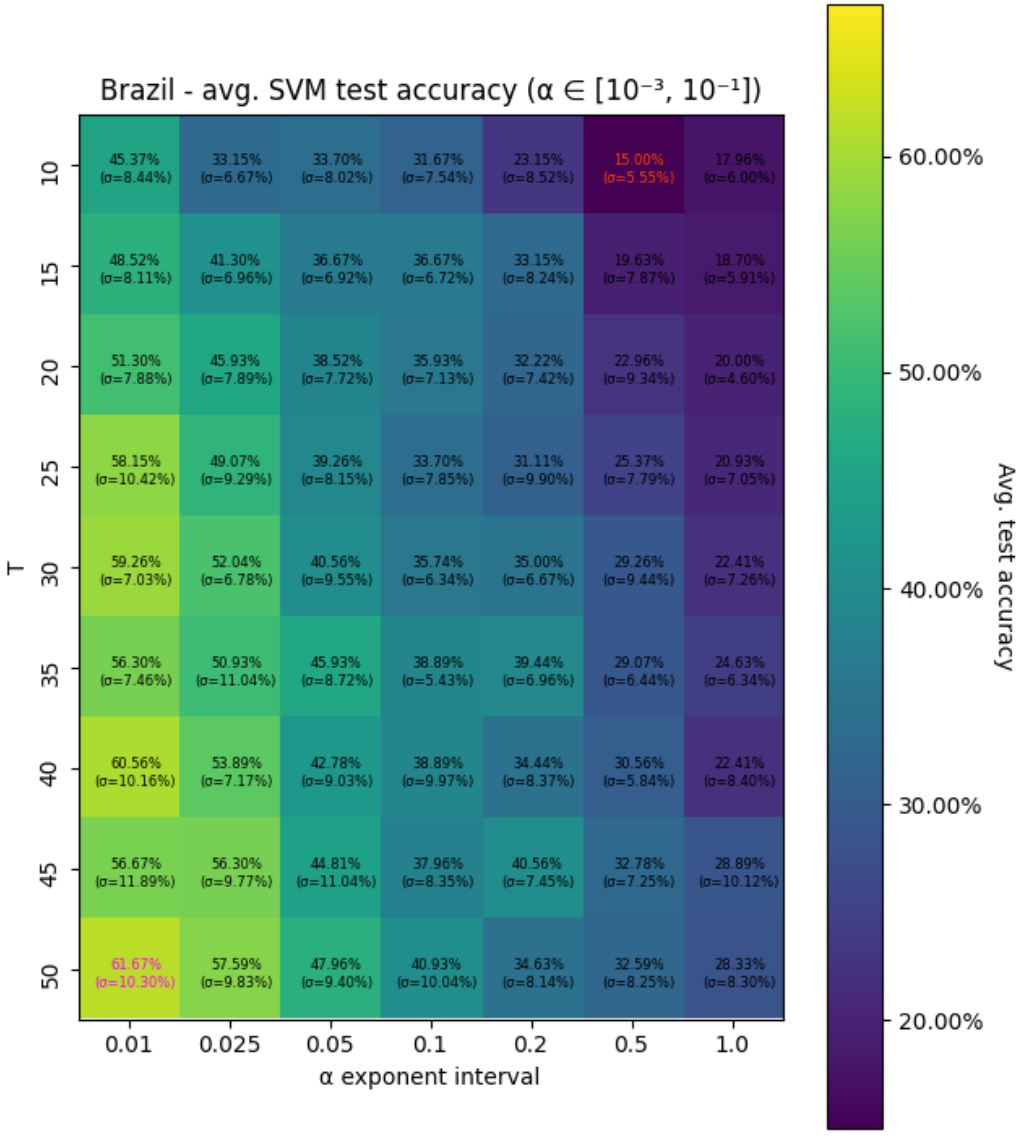


Figure A.10: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - SVM - Brazil airports

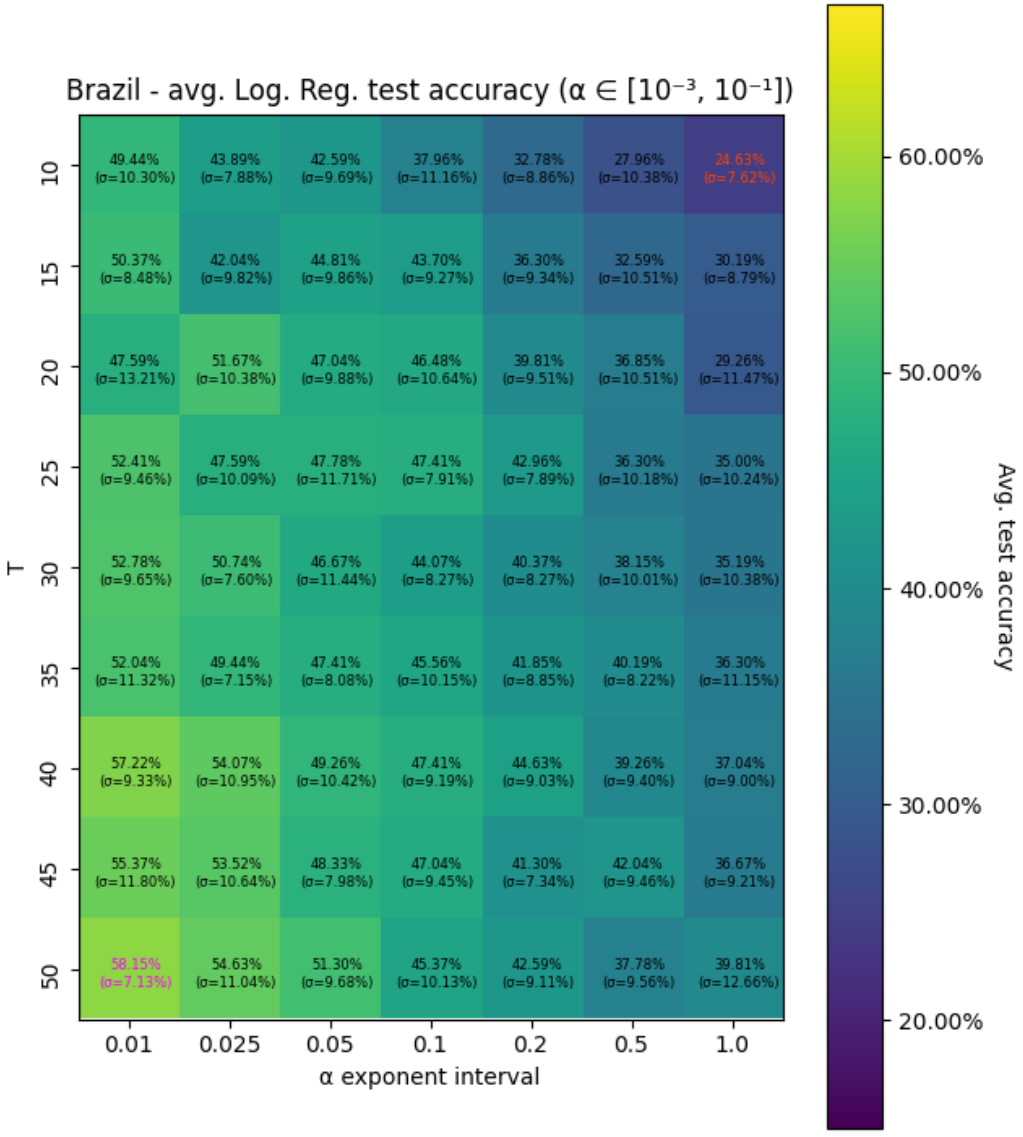


Figure A.11: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - Logistic Regression - Brazil airports

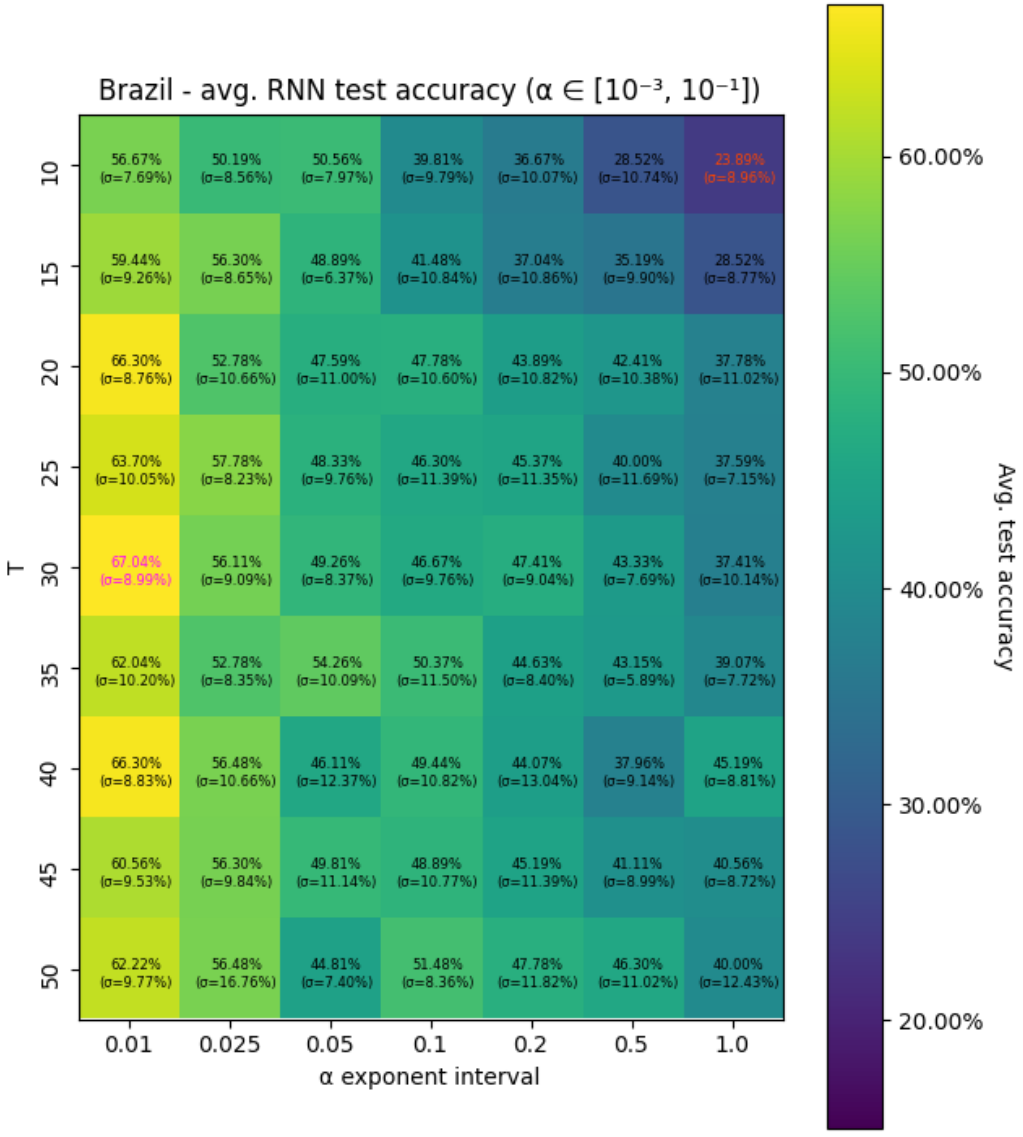


Figure A.12: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - RNN - Brazil airports

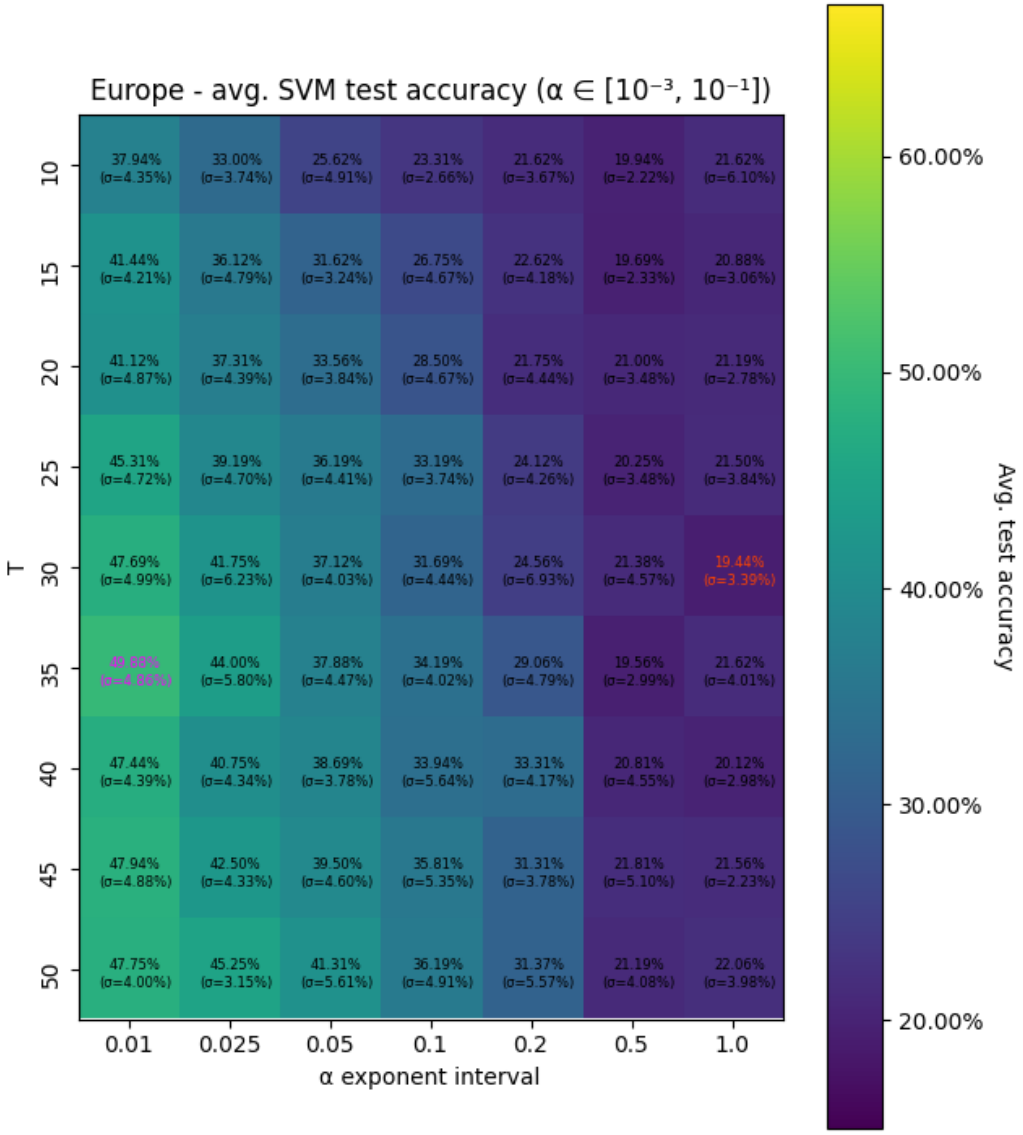


Figure A.13: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - SVM - Europe airports

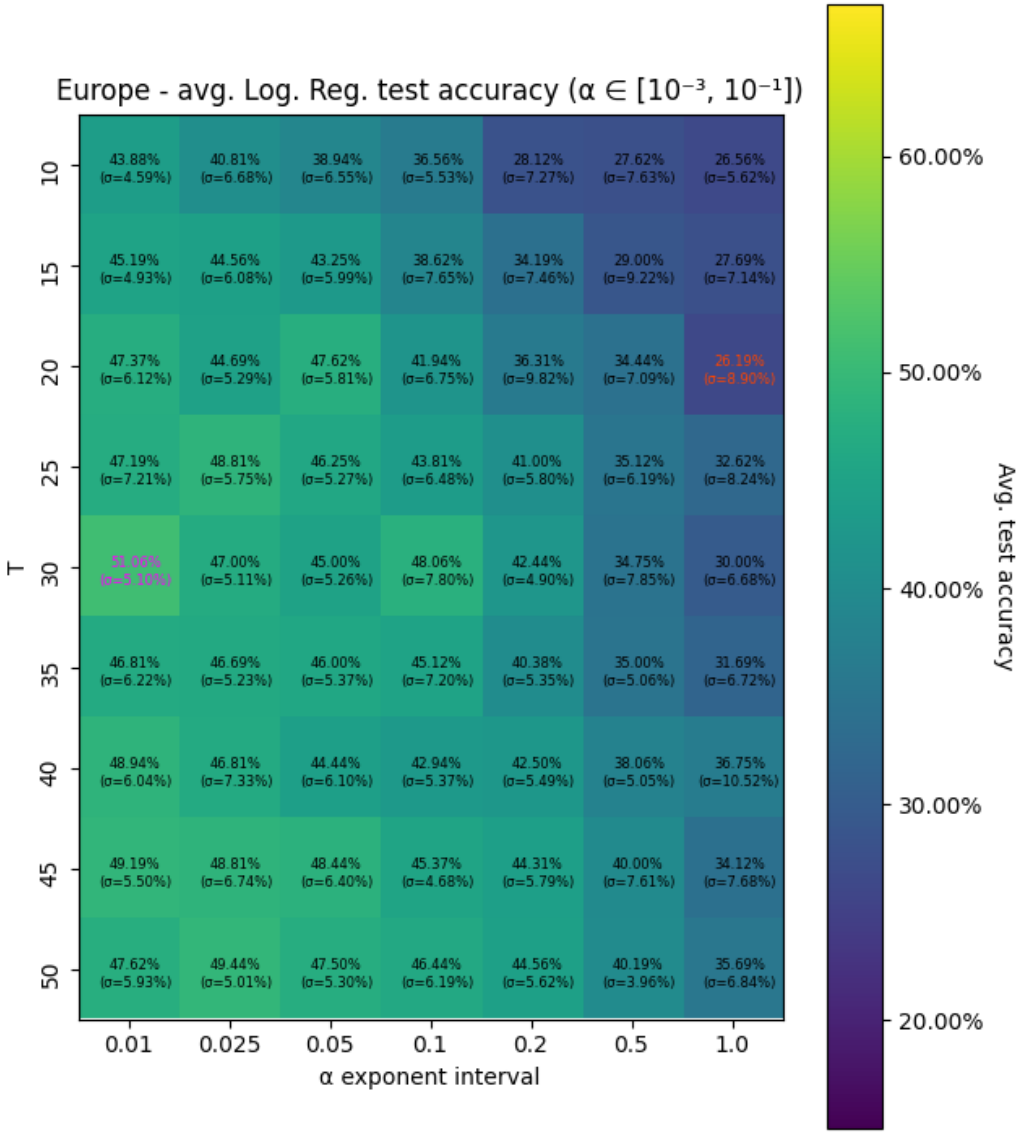


Figure A.14: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - Logistic Regression - Europe airports

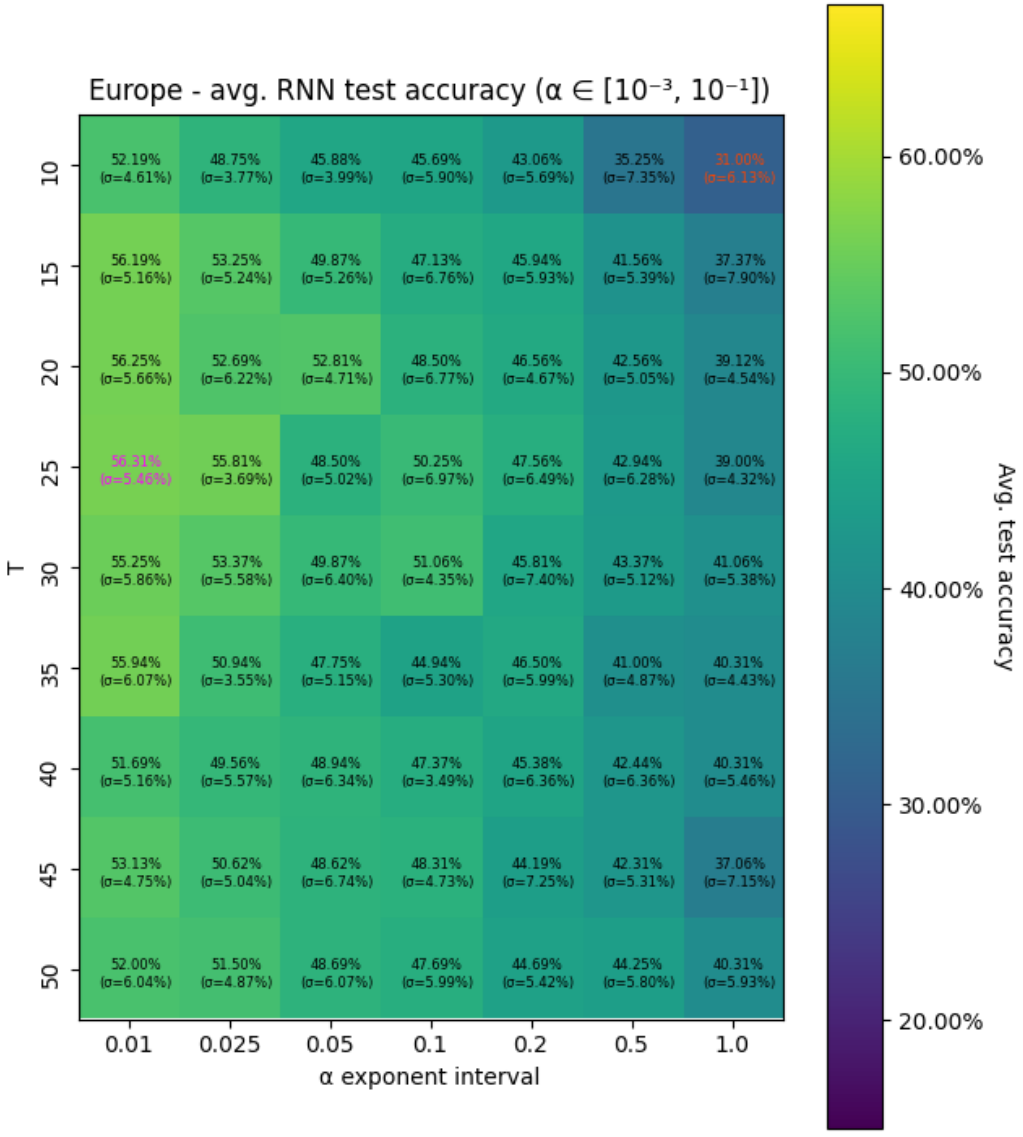


Figure A.15: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - RNN - Europe airports

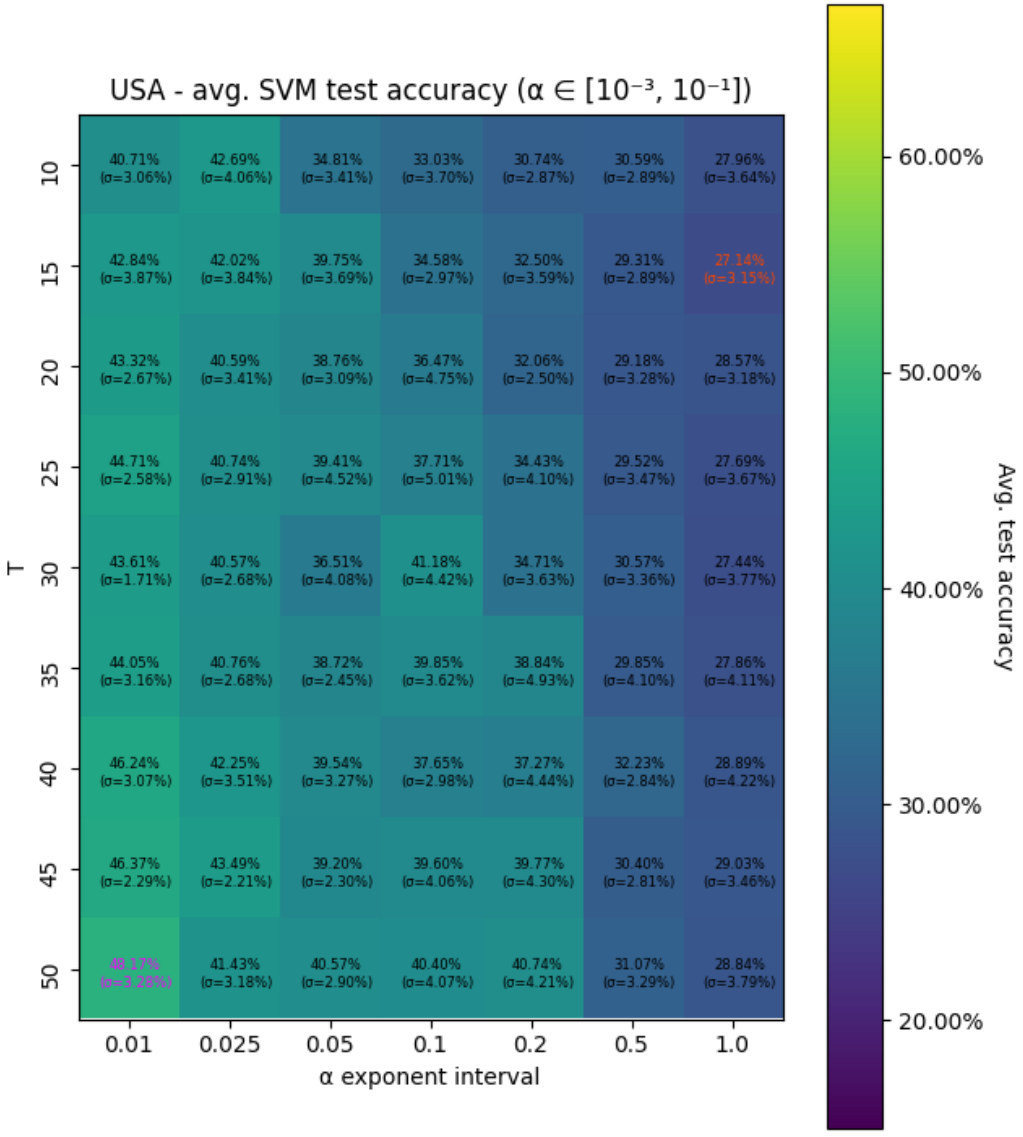


Figure A.16: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - SVM - USA airports

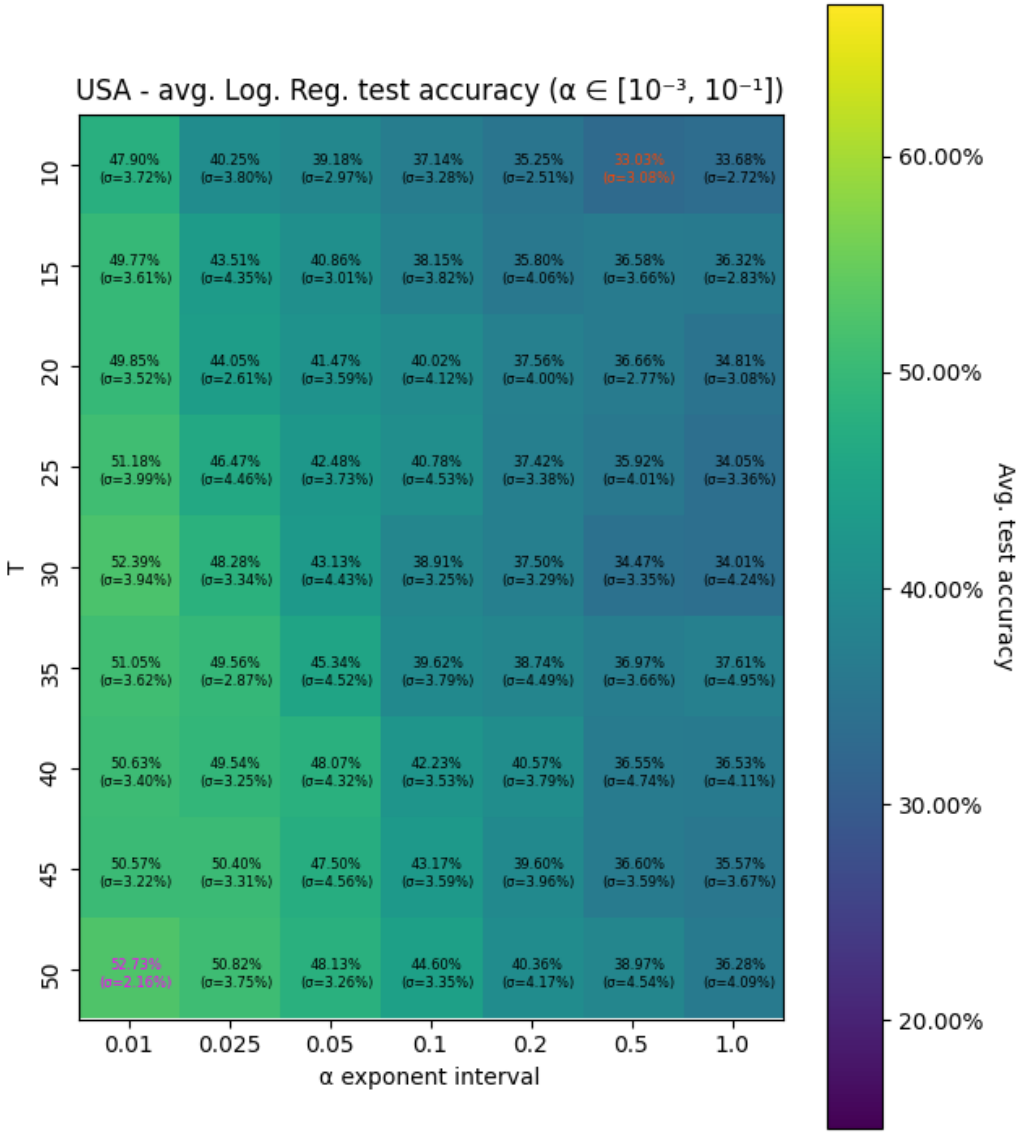


Figure A.17: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - Logistic Regression - USA airports

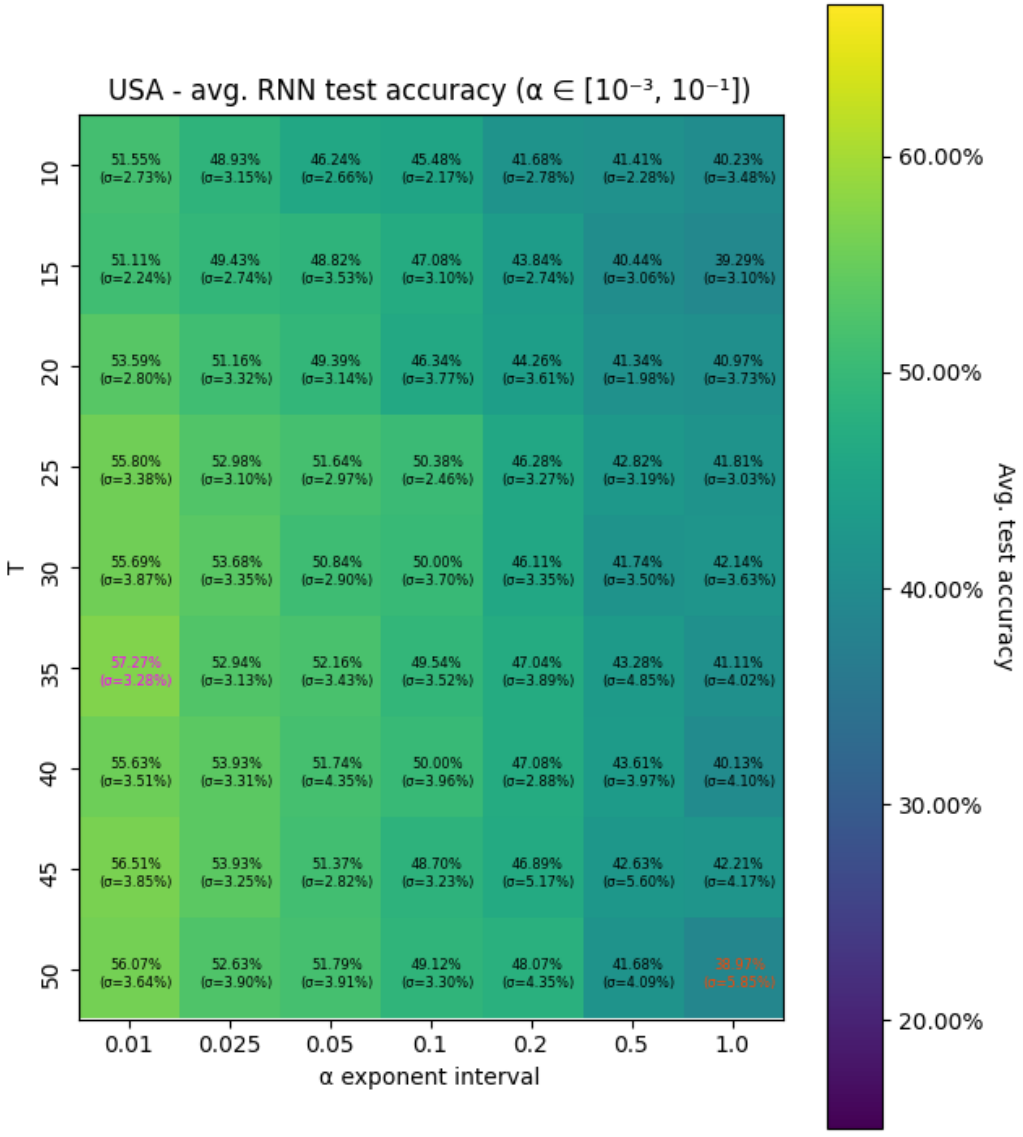


Figure A.18: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-3}, 10^{-1}]$) - RNN - USA airports

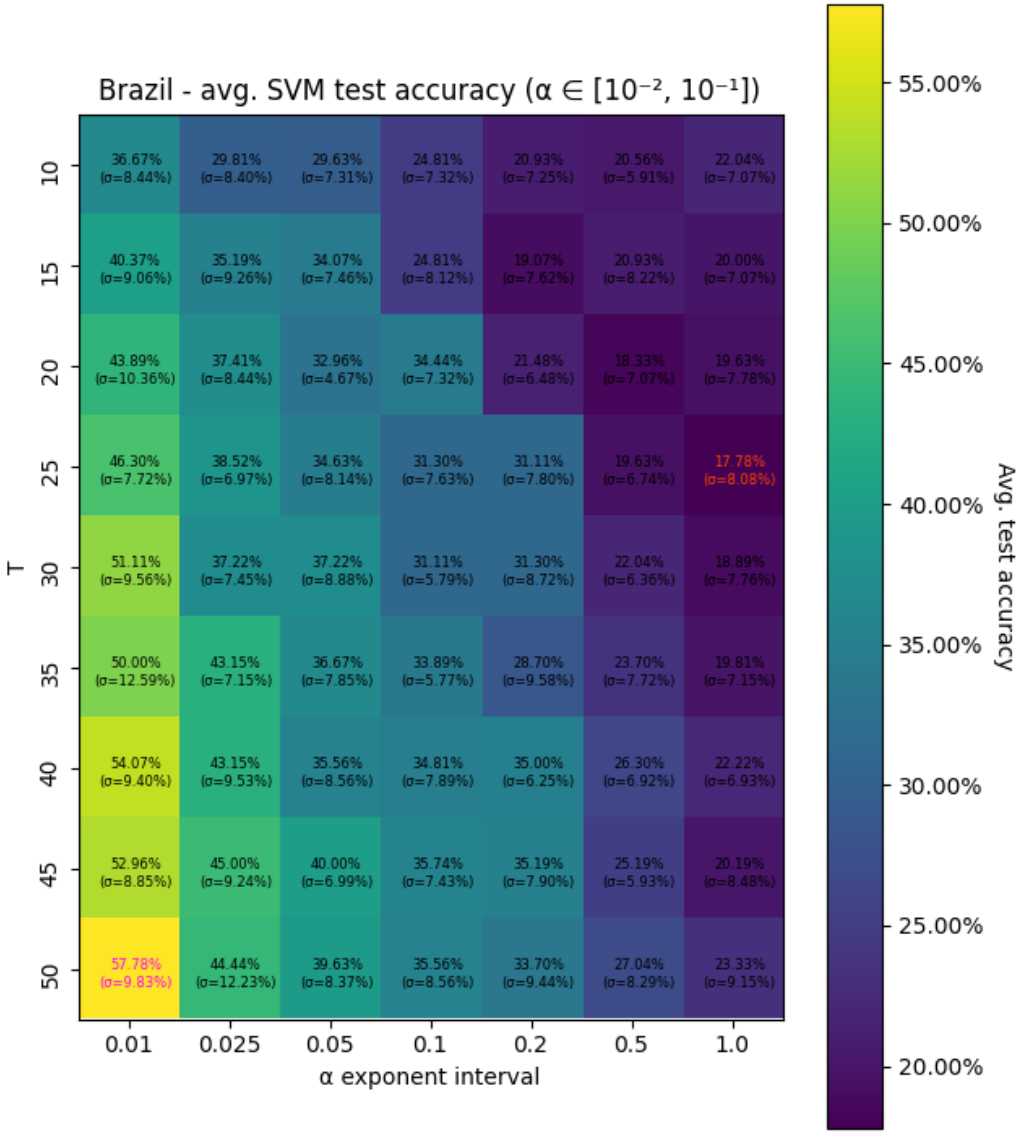


Figure A.19: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - SVM - Brazil airports

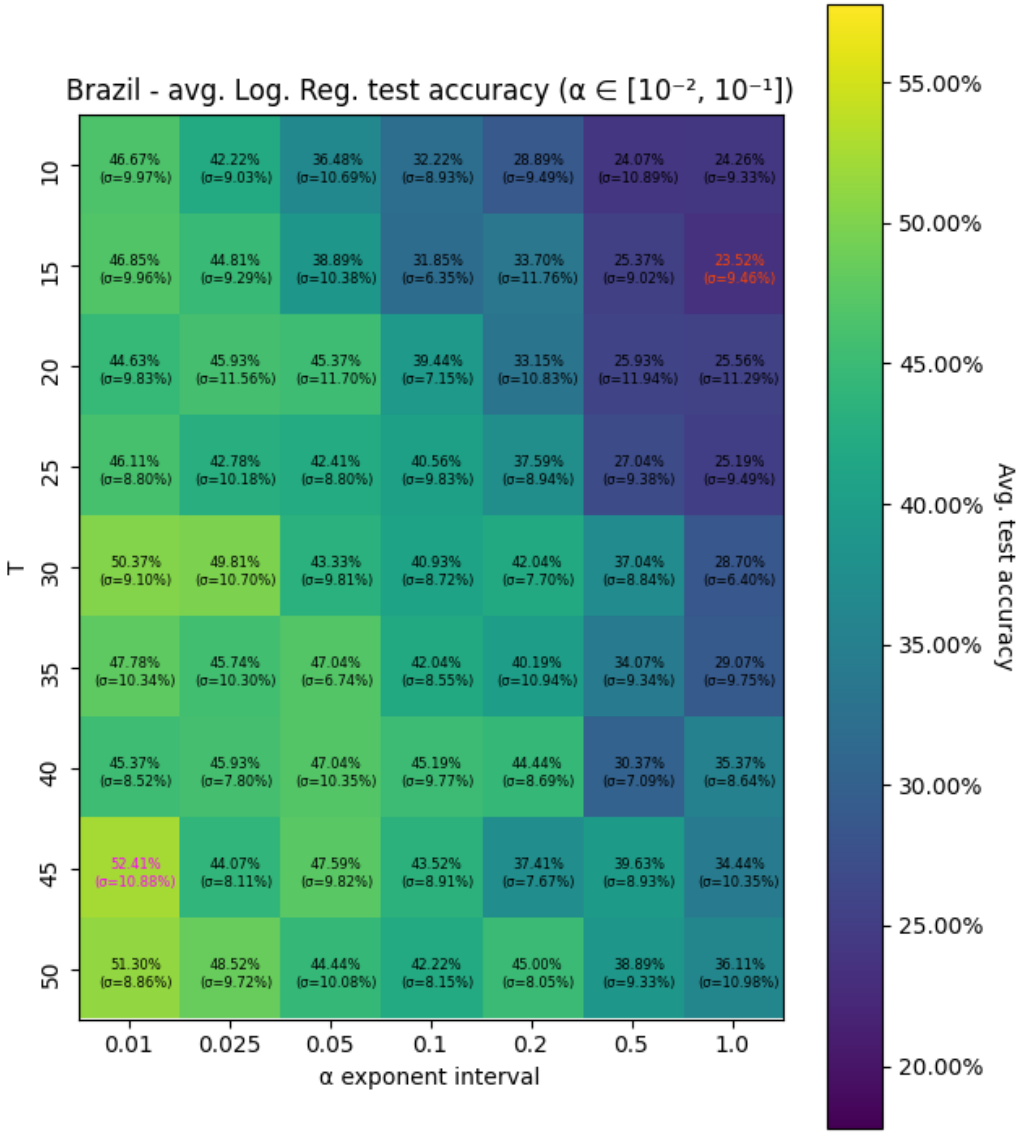


Figure A.20: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - Logistic Regression - Brazil airports

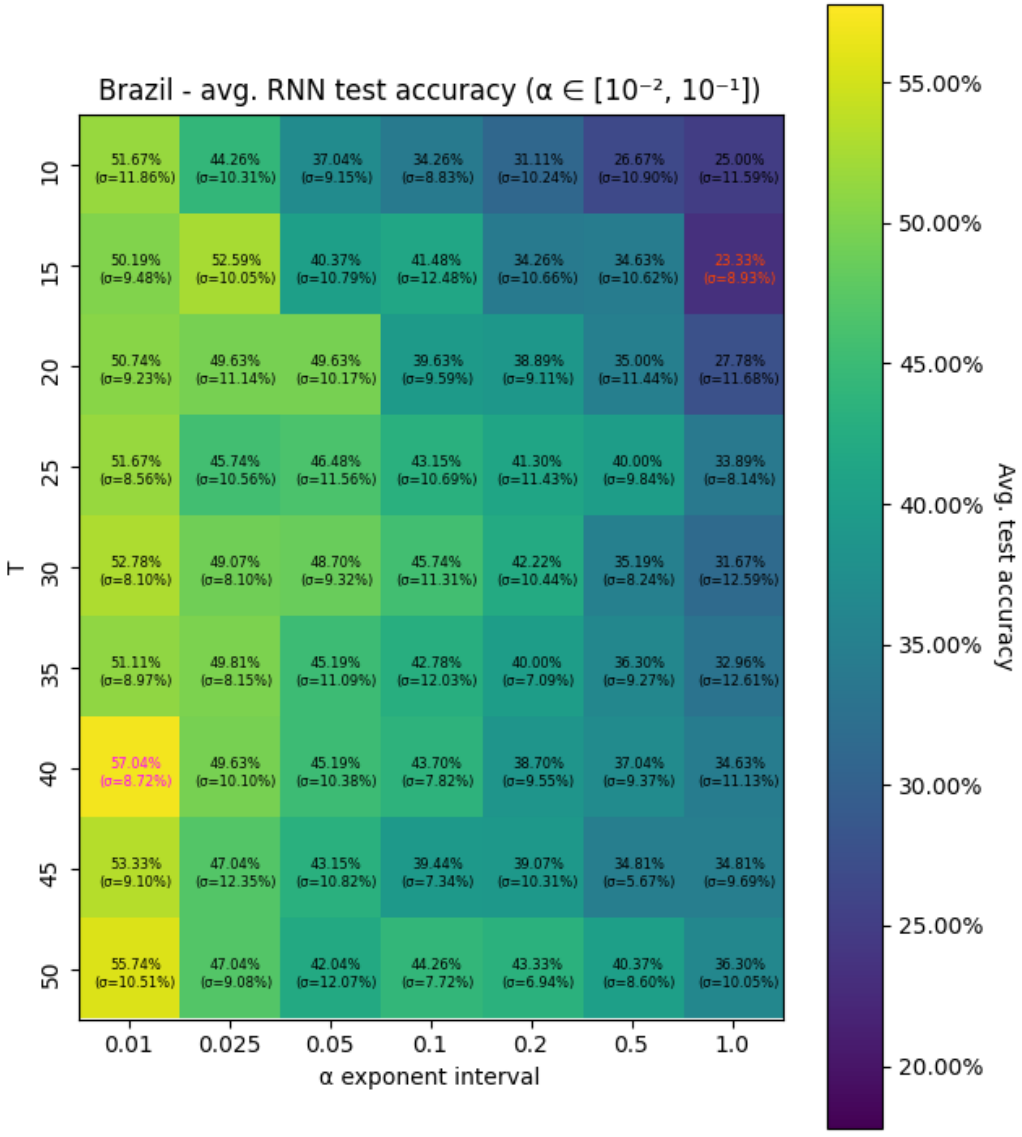


Figure A.21: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - RNN - Brazil airports

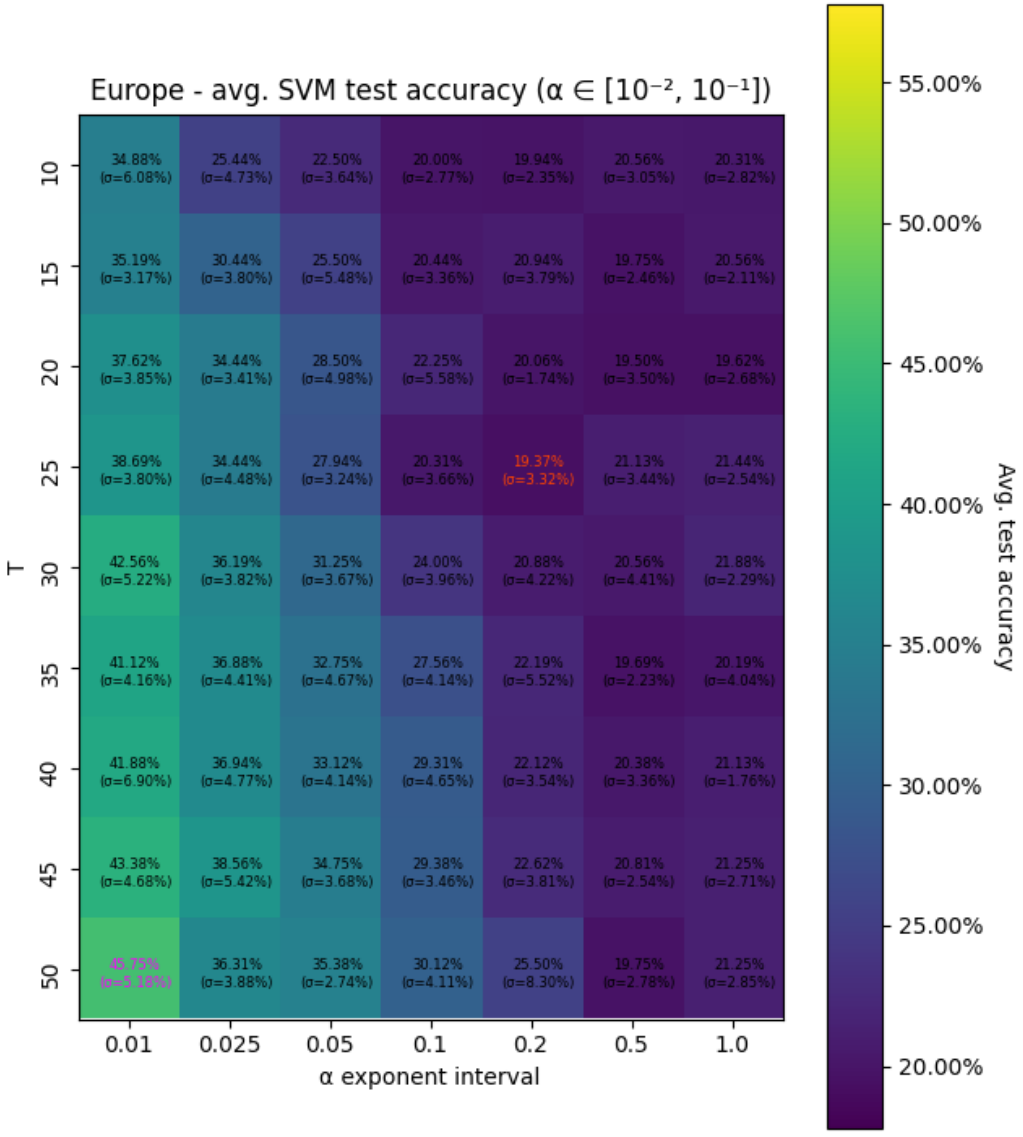


Figure A.22: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - SVM - Europe airports

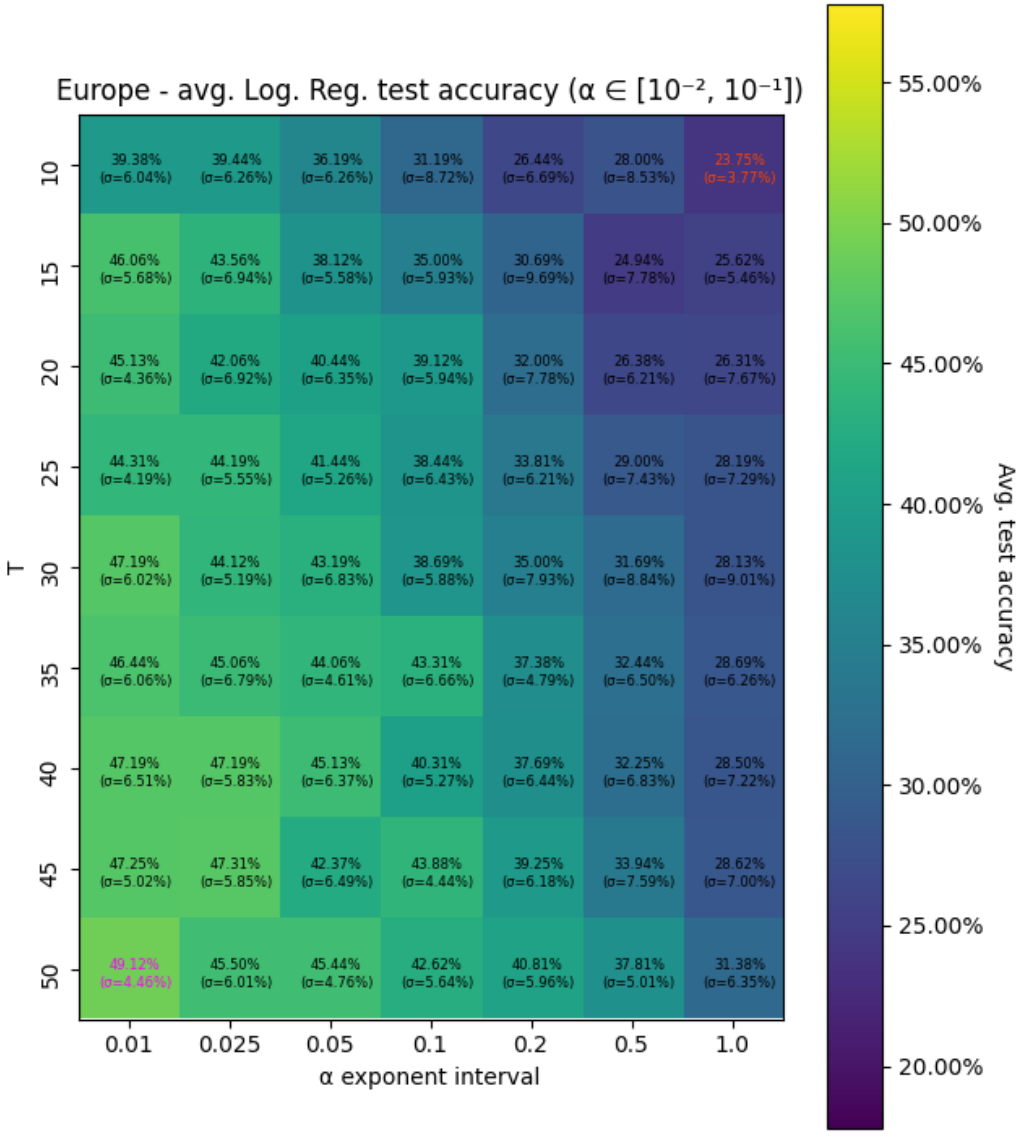


Figure A.23: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - Logistic Regression - Europe airports

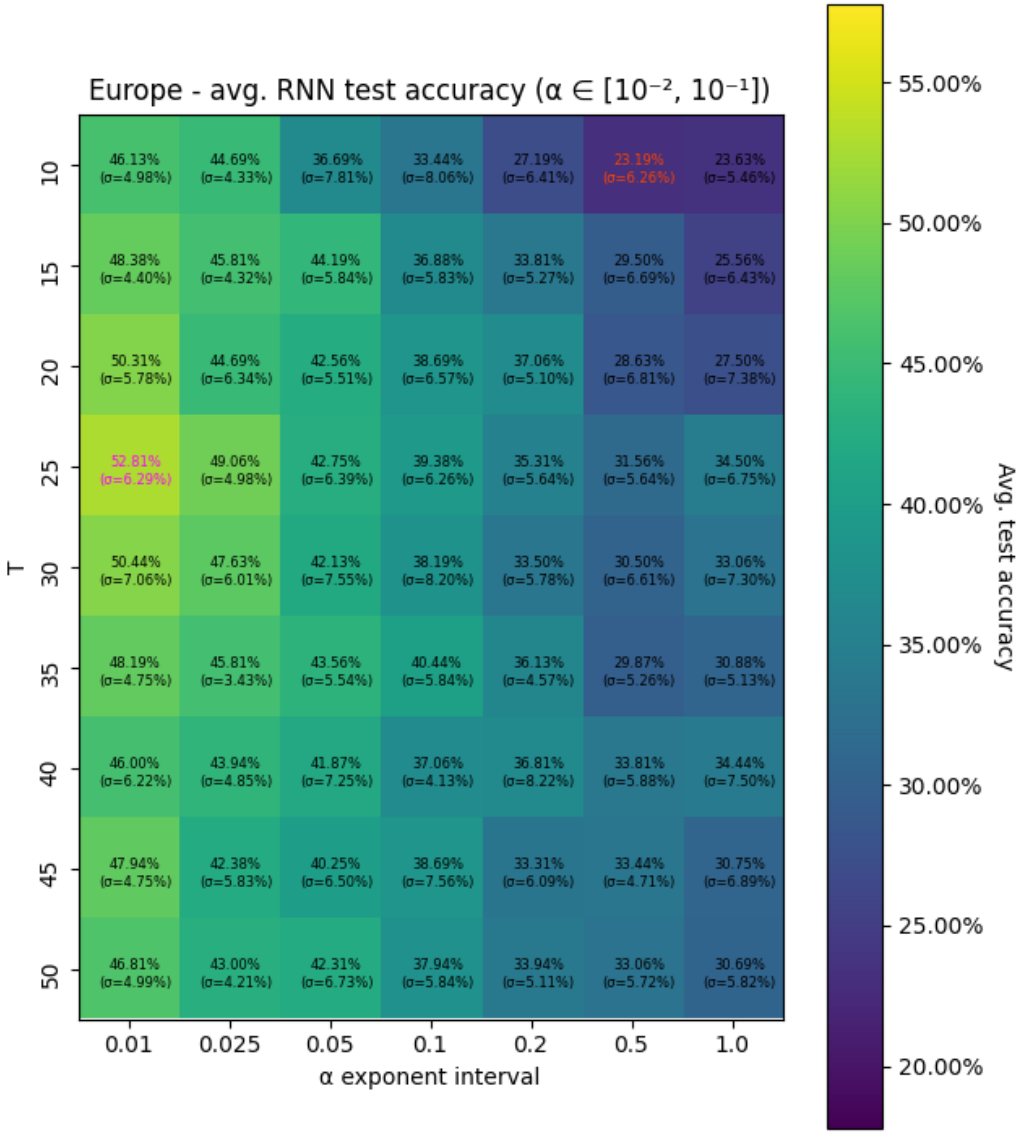


Figure A.24: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - RNN - Europe airports

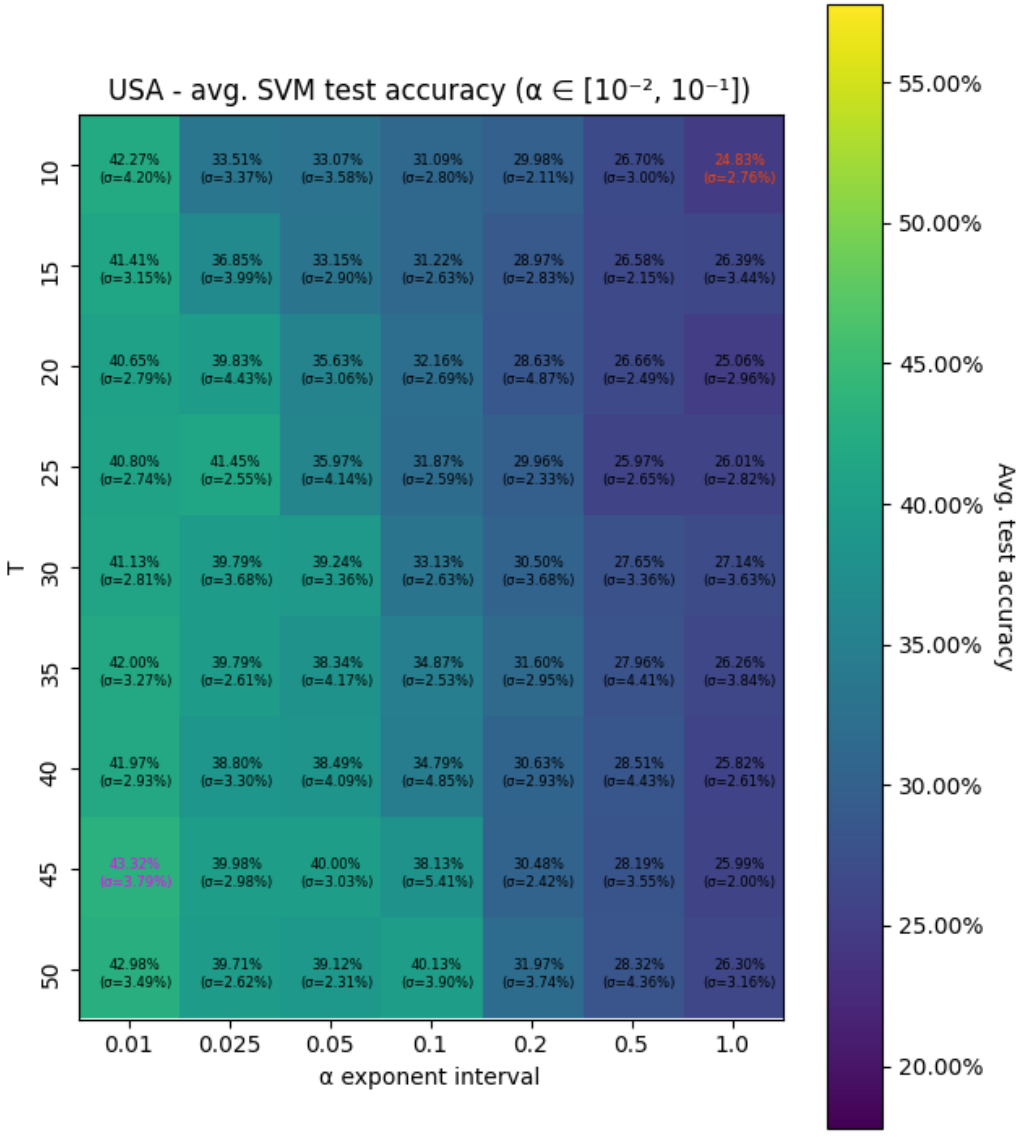


Figure A.25: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - SVM - USA airports

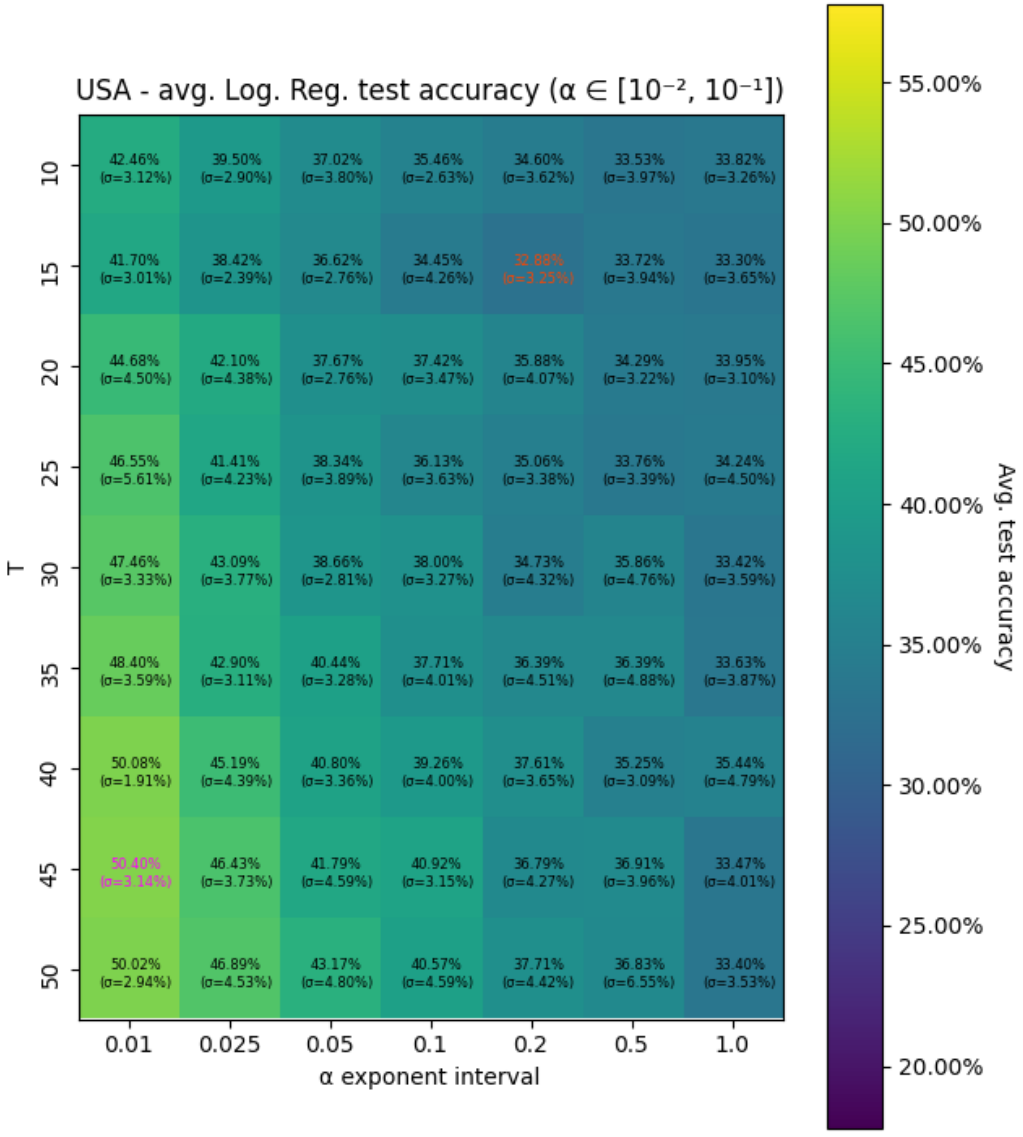


Figure A.26: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - Logistic Regression - USA airports

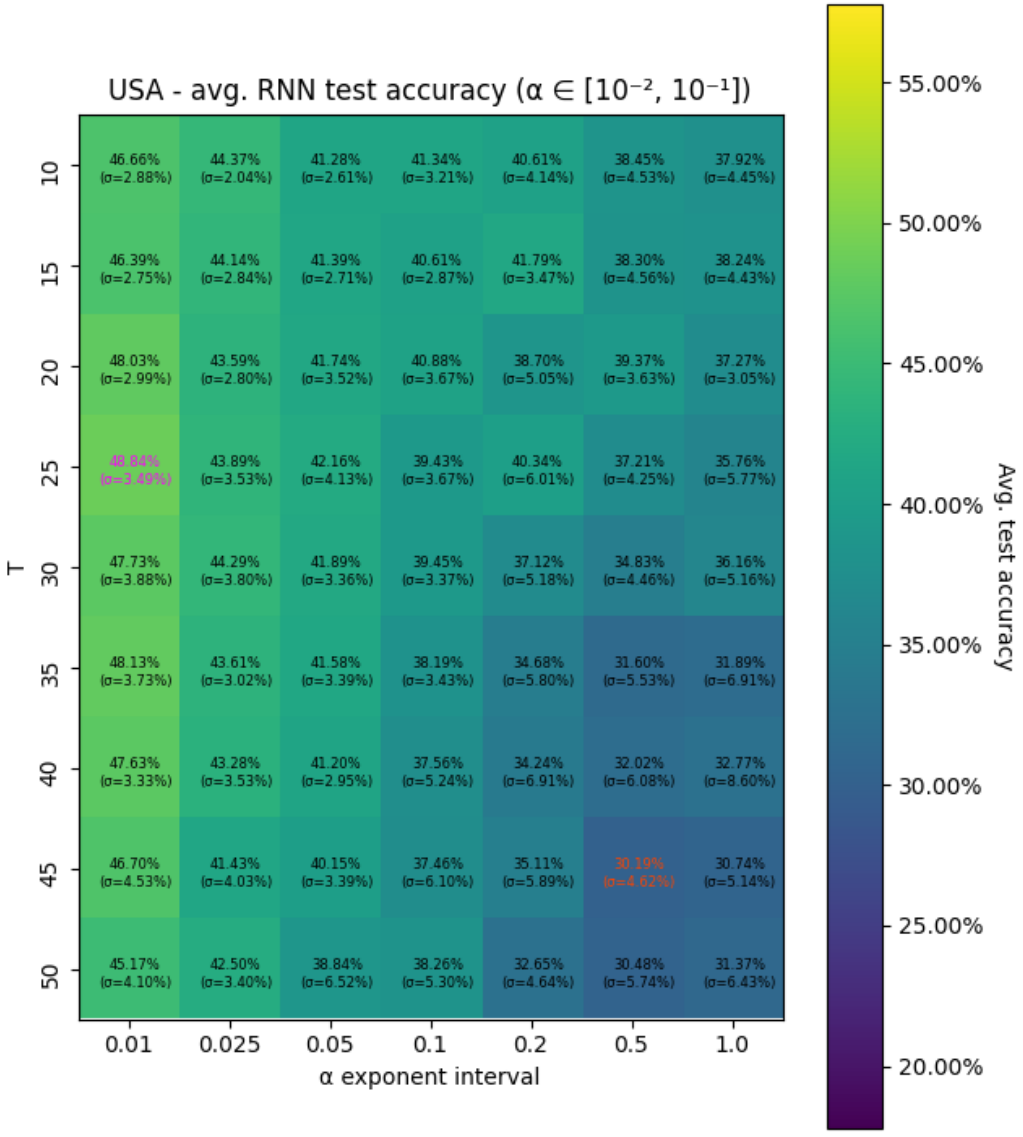


Figure A.27: Average test accuracy of 20 executions per combination of parameters ($\alpha \in [10^{-2}, 10^{-1}]$) - RNN - USA airports