Node Regression using Graph-Based SVM Kernels

1 INTRODUCTION

In this work, we present a methodology in which we use graph-based kernels to involve the support vector machine (SVM) technique and predict the node values of a graph. In more detail, we apply kernel computation using the Propagation algorithm, and then we used a multi-output SVM regressor in order to provide us with the node labels. We make experiments with two datasets. First, we use the parking violation data of Thessaloniki's controlled parking system and we predict the parking violation rate of sectors. Then, we used the chickenpox dataset and tried to predict disease outbreaks. Finally, we make additional experiments for both datasets using common SVR kernel methods as well as the K-Nearest Neighbor algorithm.

2 DATA

In this section, we describe the two datasets and we mention their shape.

2.1 Parking Violation Dataset

This dataset is based on the history of municipal police checks and consists of 27 features. Our target value is the parking violation rate measured for a sector at a point in time. In more detail, our source dataset is a list of 5074 data frames each dataframe describes a graph for a time step. Every dataframe has 222 rows that represent the parking sectors and the features are described by the columns. You can get more information about features and feature selection process in our of our previous works [2]. A lot of our data points is artificially created using smoothing and data augmentation techniques. For this reason we have the necessary information that describes, if a data point is raw in order to evaluate using only raw targets. Also we have a dataframe that describes graph edges and edge weights.

2.2 Chickenpox Dataset

A dataset of county level chicken pox cases in Hungary between 2004 and 2014 [3]. The underlying graph is static - vertices are counties and edges are neighbourhoods. Vertex features are lagged weekly counts of the chickenpox cases (we included 4 lags). The target is the weekly number of cases for the upcoming week (signed integers). Our dataset consist of more than 500 snapshots (weeks). In more detail, our source dataset is a list of 507 dataframes each dataframe describes a graph for a snapshot (week). Every dataframe has 20 rows that represent the cities and the features are described by the columns. Also we have a dataframe that describes graph edges and edge weights. Table 1 describes the source data shapes for both used datasets.

Table 1: Source Data Shapes

Data	Parking dims.	Chickpox dims.
X	(5681,222,28)	(507,20,4)
y	(222,5681)	(20,507)
Edges Weights	(1689,3)	(41,3)
Evaluation Mask	(222,5681)	None

3 DATA PREPROCESSING

In this section, we describe the process we apply in order to transform both datasets into a suitable format for our models.

3.1 Parking Data Preprocessing

Our data is already in the appropriate format as standardization has been applied to the features and the edge weights are normalized in the interval $(0.1 - 0.9)$. The only processing we have applied to our data has to do with changing its format so that it is suitable as input to the model we propose in this approach. In more detail, our data was transformed into a list of 5681 graphs each of which described by a list of 3 items. The first item is a set that constitutes the edges, the second is a dictionary constituting the features and the third is also a dictionary describing the edge weights. Table 1 shows the data shapes after transformation.

Table 2: Final data shapes

Data	# dims.
X	(5681,3)
y	(5681,222)
Edges Weights	(1689,3)
Evaluation Mask	(5681,222)

Next, we would like to describe in more detail the second dimension of X which is a composite dimension and the number 3 does not denote numerical values but a list of 3 dimensions. Table 3 describes in detail the format and length of the list elements that describe a graph.

Table 3: Description of the second dimension of X

Information	format	length
Set with edges	(e_1, e_2)	1689
Dict with features	$e_1 : [f1, f2, ..., f28]$	222
Dict with edges weights	$(e_1, e_2) : w$	1689

Finally we divide our dataset in order to split it in train and test sets, In more detail after splitting train-set is consisted of 5074 graphs and test set of 609 graphs.

3.2 Chickenpox Data Preprocessing

Above we describe in detail the procedure we followed to convert the parking dataset, however in this chapter we do not describe in detail the process for Chickebox because it is exactly the same. Briefly, we divide our dataset in order to split it in train and test sets, In more detail after splitting train-set is consisted of 403 graphs and test set of 104 graphs.

4 SVR MODEL

In this section we describe the process we followed to create one graph-based multi-output SVM regressor model to predict parking violation rate and disease outbreaks.

4.1 Graph Based Propagation Kernel

We use the graph kernels method proposed by [4]. They are based on the idea of propagating label information between nodes of the graph, based on the graph structure. A graph is considered to have attributes on nodes, where in the case of labels they correspond to One-Hot-Vectors of the full dictionary of labels. In general, the algorithm applies kernel computation between all input graph combinations, thus it accepts as input a list of graphs information and outputs and an adjacency matrix with shape (N_{graphs}, N_{graphs}). We apply the above technique by using the PropagationAttr algorithm from Grakel library [1], while we change the final kernel computation function. In Fig. 1 we describe at a high level the application of PropagationAttr process between 2 graphs.

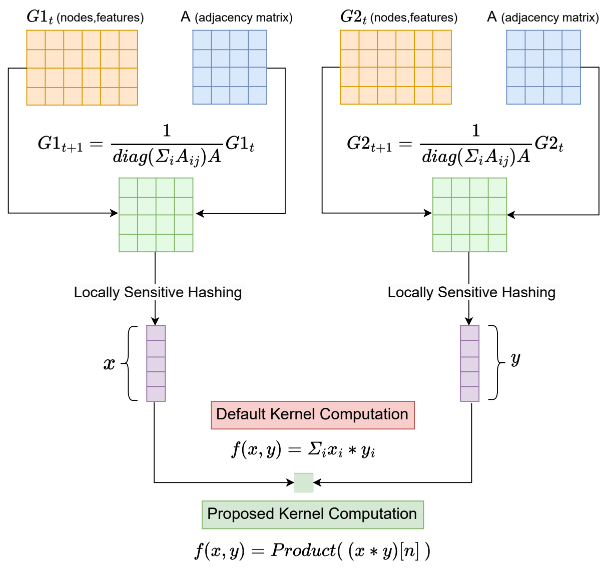
$G1_t$ (nodes,features)
A (adjacency matrix)
$G2_t$ (nodes,features)
A (adjacency matrix)
$G1_{t+1} = \frac{1}{diag(\Sigma_i A_{ij})A} G1_t$
$G2_{t+1} = \frac{1}{diag(\Sigma_i A_{ij})A} G2_t$
Locally Sensitive Hashing
Locally Sensitive Hashing
x
y
Default Kernel Computation
$f(x,y) = \Sigma_i x_i * y_i$
Proposed Kernel Computation
$f(x,y) = Product(\ (x*y)[n]\ )$

Figure 1: Multi-output graph-based SVM kernel Regression

[1]https://ysig.github.io/GraKeL

More information for the calculation of Local Sensitive Hashing (LSH) could be found at [1]. At this point we would like to focus on the kernel computation function $f(x, y)$. We replace the default function by applying a convolution-based approach. Where x, and y are the vectors produced from LSH for each graph. We treated the vectors as signals and calculated the convolution between them. Then we calculated the convolution product, which is only given for points where the signals overlap completely.

4.2 Multi-output SVM Regressor Model

As a first step, we gave as input to the PropagationAttr model the list of graphs that made up the train set, we applied fit and transform to them and then we applied transform to the graphs of the test set. So we had 2 adjacency matrices, one for the train graphs and one for those of the test-set. We then used a multi-output SVM Regressor from the Sklearn library [2]. In this model, we gave as input the adjacency matrix of the train-set with shape (N_{graphs}, N_{graphs}) and as output, we gave train targets with shape $(N_{graphs}, N_{sectors})$. We apply training with a pre-computed kernel so that it does not apply kernel computations since they have been previously applied by PropagationAttr. This is how we trained our model. Then we gave as input the adjacency matrix of the test graphs and took as output the predictions for the test-set.

5 EXPERIMENTS ON PARKING DATASET

In this chapter, we perform several experiments for Parking dataset to optimize the choice of hyper-parameters and then compare our model with other approaches.

5.1 Hyper-Parameters Tuning

We performed experiments with several combinations of hyper-parameters to choose the combination that most minimizes the prediction error. The parameters that PropagationAttr accepts as arguments are as follows:

- t_{max}: Maximum number of iterations.
- W: Bin width. The bin is a parameter that affects the calculation of locally sensitive hashing between nodes.
- M: The preserved distance metric (on local sensitive hashing)

Table 4 show us the calculated error metrics on test set during different experiments. We use Mean Absolute Error (MAE) and Mean Squared Error (MSE) metrics. It is very important to mention that we used the test-mask so that the metrics only concern comparisons made between predictions and actual targets.

Looking at the results we notice that the variation of the error during the experiments is very small but this is reasonable since the range in our target is small $(0-1)$. Finally our best model have fitted with t_{max} = 10, W = 20, and M = L2. Table 5 describes the influence of the proposed convolution function for kernel computation. It compares 2 versions of the SVM PropagationAttr kernel. First is the one in which the default kernel computation function was used, while the second is the one in which proposed function was used.

[2]https://scikit-learn.org

Table 4: Experiments using different hyper-parameters of PropagationAttr (training time is described in minutes).

t_{max}	W	M	MAE	MSE	Training Time
5	20	L1	0.19055	0.07007	18
5	25	L1	0.19001	0.07027	16
5	30	L1	0.19321	0.07354	15
10	20	L1	0.19021	0.07054	22
10	25	L1	0.18928	0.07071	20
10	20	L1	0.19328	0.07172	18
5	20	L2	1.9158	0.07102	19
5	25	L2	0.19064	0.07041	17
5	30	L2	0.19041	0.06985	16
10	**20**	**L2**	**0.18836**	**0.06975**	**23**
10	25	L2	0.19318	0.07105	19
10	30	L2	0.19018	0.07064	19

Table 5: Influence of proposed kernel computation function

Model	MAE	MSE	T. Time
SVM Propagation kernel	0.18896	0.069753	16
SVM Propagation Conv kernel	**0.18836**	**0.069753**	**23**

5.2 Comparison with usual SVM Kernels

After we finished optimizing the hyper-parameters we wanted to compare our work with some of the usual non-graph-based kernel methods. For this reason, we solved the problem as a simple regression task, without the information on the graphs (edges and edge weight). The methodology we followed is as follows. First, we applied Randomized Search 4-fold cross-validation with 4 different kernels (rbf, linear, sigmoid, poly) and various combinations of hyper-parameters. At this point we want to mention that we used the half training set only for Randomized Search, because using the whole training set was quite time consuming. We found the kernel that minimizes the error the most and then compared this model with our approach. Table 6 shows the results of 4-fold cross-validation while negative mean absolute error is used as a metric. Table is also sorted from the best model to the least efficient.

Analyzing the results, we understand that the RBF kernel solves our problem better compared to the rest, so the best model is the one with kernel = rbf, Tolerance = 0.01 and Gamma = auto. We trained this model and tested using raw test set to compare it with the model we propose in the present work, this comparison is shown in the Table 7. In addition the results obtained using K-Nearest Neighbor (KNN) algorithm are also shown.

In Fig. 2 we plot the actual target of test set compared to the predictions of our model while Fig. 3 shows the actual target of test set compared to the predictions of RBF model. We want to mention that we have published two neural network-based approaches to solving the present problem. Table 8 describes the comparison of the present work with our previous 2 methods.

Table 6: Experiments with different kernels (training time is described in minutes).

Kernel	Tolerance	Gamma	Neg. MAE	T. Time
rbf	**0.01**	**scale**	**-0.100696488585**	**51**
rbf	0.01	auto	-0.1007046022164	47
rbf	0.001	auto	-0.1010836202412	40
rbf	0.001	scale	-0.1011068083982	40
poly	0.001	auto	-0.1057053211328	37
poly	0.001	scale	-0.1057675000595	35
poly	0.01	auto	-0.1058413358572	51
poly	0.01	scale	-0.105932198324	51
linear	0.01	scale	-0.1095400208092	159
linear	0.01	auto	-0.1095400208092	152
linear	0.001	scale	-0.1095400208092	136
linear	0.001	auto	-0.1095400208092	132
sigmoid	0.01	auto	-852.333941108	61
sigmoid	0.001	auto	-852.334019555	63
sigmoid	0.01	scale	-852.334019555	62
sigmoid	0.001	scale	-852.334019555	71

Table 7: Experimental comparison between proposed method and RBF-kernel, KNN

Model	MAE	MSE	T. Time
SVM Propagation Conv kernel	**0.18913**	**0.069602**	**23**
RBF Kernel	0.20653	0.08223	80
101-Nearest Neighbor	0.20062	0.07706	3
Evaluation on Training set			
SVM Propagation Conv kernel	**0.7432**	**0.01052**	23
RBF Kernel	0.08165	0.01449	80
101-Nearest Neighbor	0.08085	0.01412	70

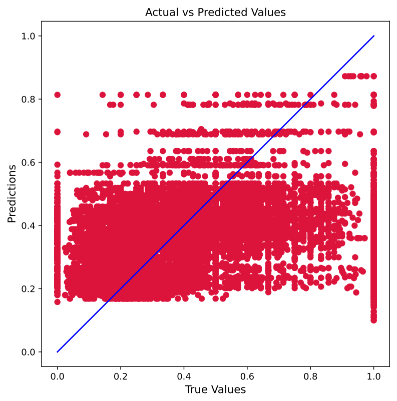

Figure 2: Multi-output graph-based SVM kernel Regression (Parking Dataset)

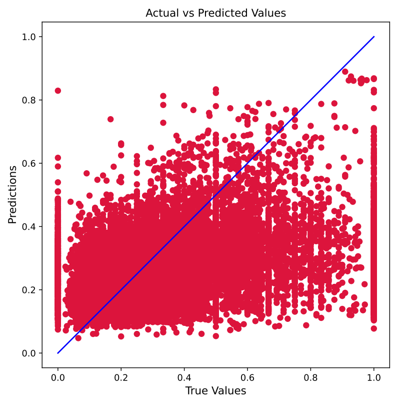

Figure 3: RBF Kernel Regression (Parking Dataset)

Table 8: Experimental comparison of the proposed method with our previous works.

Model	MAE	MSE
DNN	0.1991	0.0681
SVM Propagation Conv kernel	0.1891	0.0696
TGCN + LSTM	**0.1662**	**0.0468**

6 EXPERIMENTS ON CHICKENPOX DATASET

In this chapter, we perform several experiments for the Chickenpox dataset to optimize the choice of hyper-parameters and then compare our model with other approaches. We do not show all the results for the hyper-parameter optimization experiments in order not to make the report long. However, the results are visible in the code.

6.1 Hyper-Parameters Tuning

We applied the same methodology that we used above for the parking dataset. We do various experiments to optimize hyper-parameters of PropagationAttr. Finally our best model have trained with t_{max} = 15, W =25, and M = L2. The calculated metrics on test set was MAE = 0.12364 and MSE = 0.02792.

6.2 Comparison with usual SVM Kernels

After we finished optimizing the hyper-parameters we wanted to compare our work with some of the usual non-graph-based kernel methods. We tested various kernels through a grid search. Finally, the best model was fitted with kernel = RBF, tol = 0.001, and gamma = auto. The calculated metrics on test set was MAE = 0.11455 and MSE = 0.02227. Table 2 summarizes the results of various kernels tested for the Chickenpox dataset. In addition the results obtained using K-Nearest Neighbor (KNN) algorithm are also shown.

Table 9: Chickenpox Experimental comparison between proposed method and RBF-kernel, KNN (training time is described in seconds)

Model	MAE	MSE	T. Time
SVM Propagation Conv kernel	0.12324	0.02632	24
RBF Kernel	**0.11455**	**0.02227**	4
101-Nearest Neighbor	0.12231	0.02651	2
Evaluation on Training set			
SVM Propagation Conv kernel	0.06808	0.01134	24
RBF Kernel	0.07475	0.01081	4
101-Nearest Neighbor	**0.06336**	**0.00960**	10

REFERENCES

[1] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. 2021. A Survey on Locality Sensitive Hashing Algorithms and their Applications. https://doi.org/10.48550/ARXIV.2102.08942

[2] Nikolaos Karantaglis, Nikolaos Passalis, and Anastasios Tefas. 2022. Deep Learning for On-Street Parking Violation Prediction. (2022), 1–5. https://doi.org/10.1109/IVMSP54334.2022.9816222

[3] Benedek Rozemberczki, Paul Scherer, Oliver Kiss, Rik Sarkar, and Tamas Ferenci. 2021. Chickenpox Cases in Hungary: a Benchmark Dataset for Spatiotemporal Signal Processing with Graph Neural Networks. arXiv:cs.SI/2102.08100

[4] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. 2020. GraKeL: A Graph Kernel Library in Python. *Journal of Machine Learning Research* 21, 54 (2020), 1–5.