Multi-GNN for ovarian cancer survival prediction

Colombari Martina
196021
270396@studenti.unimore.it

Lupo Davide Rosario
189485
336936@studenti.unimore.it

Mancinelli Francesco
191738
341883@studenti.unimore.it

Abstract

Prognostication of overall survival (OS) in ovarian cancer patients is important for tailoring treatment plans and enhancing clinical decision-making. Most current state-of-the-art models only use gene expression data. There is great opportunity for improving survival prediction accuracy by incorporating multiple omics data types such as DNA methylation and copy number alteration (CNA) profiles.

This study intends to design a new framework based on Multi-Graph Neural Networks (Multi-GNN) for predicting ovarian cancer patients' overall survival data by synthesizing gene expression data, methylation data, and copy number alteration data from TCGA.

In our implementation, we adopted a novel approach to graph construction by encapsulating all gene data within individual nodes. These nodes are then interconnected using protein-to-protein interaction data sourced from STRING. We initially attempted to predict overall survival (OS) by classifying it into discrete categories, effectively binning the OS labels. After this, we transitioned to a regression model for more precise predictions. Finally, we validated our results by comparing them with two related repositories.

1 Preprocessing

1.1 Introduction

The clinical dataset was obtained from the Genomic Data Commons (GDC) Analysis Center which is an integrated data system that collects and disseminates cancer genomic and clinical data from the TCGA and TARGET cancer research programs. In this case, the platform was used to obtain an exhaustive dataset of ovarian cancer to address the research questions posed. During data collection, a set of filters was applied to maintain the consistency and quality of the genomic data. The following selection criteria were used: Program filter set to "TCGA", Project restricted to "TCGA-OV" (ovarian cancer cohort), Disease Type limited to "cystic, mucinous and serous neoplasms", and Primary Diagnosis serous cystadenocarcinoma. The data was downloaded in BCR XML format as a Clinical Supplement data type. This dataset was tailored to the research focus on serous ovarian carcinoma cases from The Cancer Genome Atlas.
Then from that dataset we filter only the user case which are registered with a OS (Overall Survival, time between the first check up and the time of death) and have data available for Gene expression, Methylation and Copy number.

1.2 Graph structure

Since we are working with graph classification, we create a graph for each user case, and each of them is structured as follows:
Each node represents a gene, incorporating associated methylation and copy number data. Nodes are connected based on the protein–protein interaction (PPI) information provided in the STRING database [1]. The edges between nodes are defined by these interactions, with edge attributes corresponding to the interaction strength values from the PPI data.
Since DNA methylation data is characterized by multiple CpG probes mapping to the same gene, we decided to integrate the methylation data inside a single node by aggregating all the information in a single value. We did this to reduce the computational complexity of the dataset, and make it possible to have more genes for the same amount of available memory, and also because averaging CpG probes mitigates the impact of technical noise, creating a more precise representation.

1.3 Preprocessing data

1.3.1 Gene expression

For gene expression we have different features available for each gene, and the following are the ones used for the creation of the dataset [2] [3] [4].

- **unstranded**: Raw read counts where strand information is ignored. Reads mapping to either DNA strand are counted toward a gene if they overlap its location.
- **tmp_unstranded** and **fpkm_unstranded**: A normalization method that adjusts for both sequencing depth and gene length, making it suitable for comparing expression within a sample.

- **fpkm_uq_unstranded**: A variation of FPKM where instead of normalizing by total fragments, you normalize by the upper quartile (75th percentile) of gene counts. This makes it more robust to highly expressed genes skewing the data.

Then normalization each feature with the application of $log_{10}(x + \epsilon)$ (ϵ is a really small value to make sure to don't have $log_{10}(0)$), and then normalize between 0 and 1 among each feature.

1.3.2 Methylation

The methylation data stored in TSV format is scrubbed and brought into a pandas data frame where probe IDs (Illumina IDs) and their corresponding beta values (0 to 1 range with 0 be unmethylated and 1 fully methylated) are retrieved. The beta values show the level of methylation for the probe. The methylation values and probe IDs are cleaned by removing missing values and casting the methylation values to floats and probe IDs to strings to preserve precision. To convert from CpG to gene level data, a conversion file with IlmnID of the probes and the gene IDs is used. We need this step because both Gene Expression and Copy Number are using the ENSG convention. For the genes with many assigned CpG probes, the methylation values for the CpG probes are simplified by determining the mean beta-value for all probes associated with that gene, thus generating a single, representative methylation score for every gene. To resolve the duplicate gene entries, the first instance is retained ensuring uniqueness. In the final step, all samples are load aligned and gene-level methylation values for the probes are min-max normalized to a [0, 1] scale.

1.3.3 Copy number

Also for Copy Number Variation we have different features available for each gene, that are: min_copy_number, max_copy_number and copy_number. According to the official documentation provided at [5], the following information is available.

- If a gene overlaps with only one segment, all three values are equal.
- If it overlaps with multiple segments:
  - min_copy_number is the minimum among all overlapping segment values.
  - max_copy_number is the maximum among all overlapping segment values.
  - copy_number is calculated as the weighted median of copy number values from all overlapped segments.
- If a gene does not overlap any segment, all three values are set to an empty string.

In our analysis, we retain only the copy_number feature, which is normalized to the range [0, 1]. Empty values are replaced with 0 before normalization.

1.4 Reduction of number of gene

We first decided to remove non-protein-coding genes, using the GTF file, and kept only the protein coding gene, so the only logically relevant for our task.
Subsequently, additional genes were removed to build a model of manageable size that fits within the available memory constraints during training.

To guide this process, we developed a ranking algorithm to select which genes to retain.
To avoid introducing bias during evaluation, we first split the dataset into training and test sets, then performed the ranking exclusively on the training data.
Division of the 2 subset was made with great attention, to make sure that the testing set was a good representation of the data, to do so we kept the same distribution, in terms of OS, and took a smaller sample, this can be observed in figure 1.

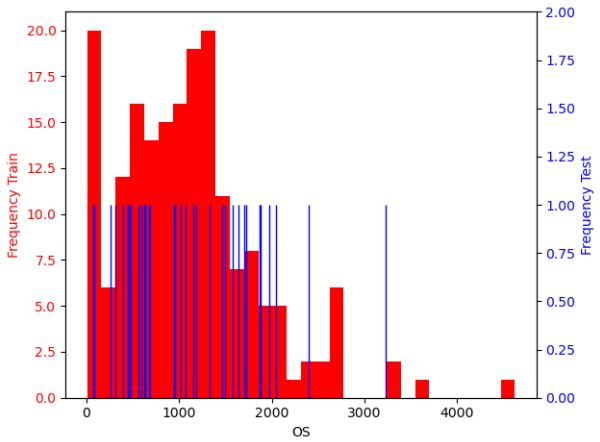


Figure 1: Distribution of Train and Test set in relation to OS.

We did not choose to split the dataset further, for the validation set, due to the limited amount of user cases available to us, we had a total of 226 distinct user.
Our initial approach involved ranking genes based on gene expression variance among all the user cases.
We selected genes with the highest variance, assuming they would carry the most information, however, this strategy favored genes with unpredictable expression patterns, which ultimately degraded model performance, particularly in classification tasks.
To improve classification, we adopted a more sophisticated scoring system, described as follows:

Compute **intra-subset variance**: For each class c, compute the variance of the gene values across all graphs in class c, then average this variance across all class subset.

Compute **inter-subset variance**: For each subset, compute the mean value of the gene, then compute variance of these means across all class subset.

$$Score_{gene} = \frac{InterVariance_{gene}}{IntraVariance_{gene}+\epsilon}$$

2 Training

During training, we followed the preprocessing approach described in the previous chapter, so we used the score based approach to choose which gene to take and still keep the same division in training and test to not introduce biases in the test sample.
First we approach the problem with classification and then with regression.

2.1 Classification

The first approach was to divide the user case in n-classes, and make a model able to classify them. This approximation helps the model improve prediction, as the model only needs to determine the range of overall survival (OS) to which each user case belongs. For the training we used a Adam optimizer [6] and ReduceLROnPlateau scheduler [7] to reduce the learning rate during training. As a loss we used a classic Cross Entropy loss. We developed many model architectures with many properties, the following are the most relevant ones, in terms of results. As input we have 6 features, 4 from gene expression, 1 from methylation and the last from the copy number.

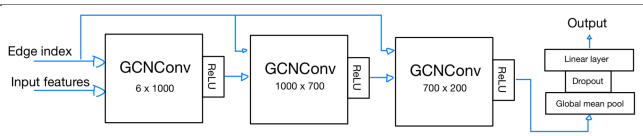
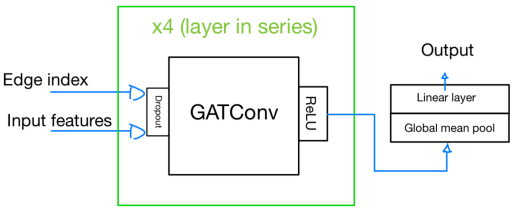
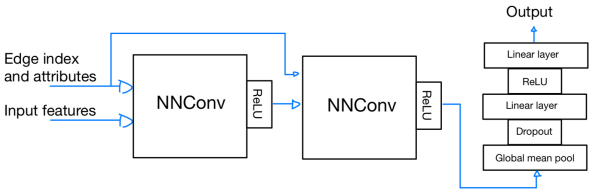
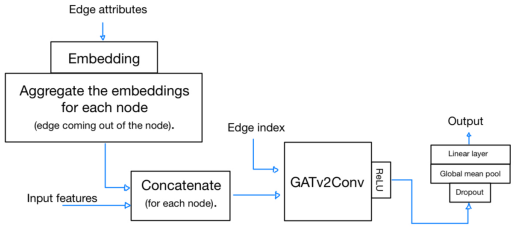
Models	Description	Scheme
"Simple_GCN"	3 graph convolution layer [8], and a linear layer at the end, which is applied right after the global mean pool [9]. This model does not take into account the edge attributes.	
"ComplexGAT"	4 GAT Layer [10] , and a linear layer at the end, which is applied right after the global mean pool [9]. This model does not take into account the edge attribute.	
"EdgeAttrGNN"	2 convolution layer, specific to take into account attribute layer [11], a 2 linear layer at the end, This model does take into account the edge attribute.	
"EdgeAttrGAT"	1 GAT convolution layers [12], and a linear classifier at the end, in this model we also apply an embedding to the edge attribute and concatenate them to the node feature which that node is from (where it start), This model does take into account the edge attribute.	

Table 1: Models

Each test is documented in the refecenced google drive [13], where datasets and model parameters are also stored.

Models	2 classes accuracy	4 classes accuracy	16 classes accuracy	Avg number of nodes
"Simple_GCN"	58%	33%	8%	6500 nodes
"ComplexGAT"	61%	32%	12%	1888 nodes
"EdgeAttrGNN"	75%	40%	14%	1329 nodes
"EdgeAttrGAT"	60%	26%	6%	1295 nodes

Table 2: Models training results

2.2 Regression

From the knowledge acquired in the classification model we build the regression, by changing the necessary component. For starters in the regression we evaluate primarily the results with the "c-index" metric [14], which is a discrimination metric that provides information about how well the model separates high-risk from low-risk patients.
This metric is applied, for this model, between every 2 samples, so it is always considered as a comparison between 2 user cases.
Since we use this metric to evaluate the model we also updated the model loss, to train the model to appropriately differentiate high-risk and low-risk patients. The new loss is a combination of 2 different functions, the first one is a custom loss function used to provide the weight's gradient based on the c-index metric results, its implementation can be observed in the figure 2.

```
175  def cindex_loss_vectorized(pred, y):
176      pred = pred.view(-1)
177      y = y.view(-1)
178
179      # Create all pairwise differences
180      # https://stackoverflow.com/questions/69797614/indexing-a-tensor-with-none-in-pytorch
181      diff = pred[:, None] - pred[None, :]
182      y_diff = y[:, None] - y[None, :]
183
184      # Mask for valid pairs where y_i > y_j
185      valid = y_diff > 0
186
187      # Apply sigmoid to the negative prediction difference
188      loss = torch.sigmoid(-diff[valid])
189
190      if valid.sum() == 0:
191          return torch.tensor(0.0, device=pred.device, requires_grad=True)
192
193      return loss.mean()
```

Figure 2: Custom loss function implementation

The second part of the loss function is a Mean Square Error loss [15] used in relation to the actual OS value; this provides stability for the training and forces the model not only to differentiate well between patients, but also to get close to the right OS value. The following is the complete loss function formula, $MSELoss() + \alpha * cindex\_loss\_vectorized()$, it also introduces another parameter α, is set to 0.1 and is used to define how much the c-index metric influences the loss function.

For the regression task we kept the "EdgeAttrGNN" model, which is the one that provided the best results for classification.

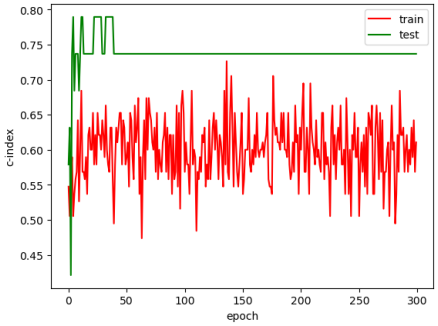


Figure 3: Regression Training

With this configuration we trained the model, in figure 3 we can observe the results for the "c-index" metric, we can see how the train set oscillates greatly for each epoch, and how the test set is stable instead.
After some investigation we found out that this irregularity happens due to the batch creation. The training batch shuffles every epoch, where the test batch is created randomly and don't change after. This impact greatly to the c-index metric.

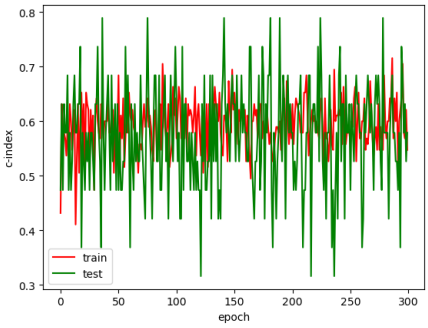


Figure 4: Regression Training

We can observe it in figure 4, where both Train and Test set randomly shuffle after each epoch.

3 Related Work

This section provides a comprehensive comparative analysis between the models proposed in our study and two leading methods from the current literature. To ensure a fair assessment, all models are evaluated on the same dataset used in our experiments, allowing for a consistent and meaningful performance comparison.

3.1 Graph Neural Networks for Graph Classification

The first *repository* [16] we considered provides implementations of a wide range of Graph Neural Network (GNN) architectures designed for graph classification tasks, including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, Graph UNet, among others. The framework also incorporates a readout module that aggregates node-level representations into a global graph embedding through pooling operations (average, max, sum), which is then passed to a fully-connected layer for classification. Finally, the evaluation process is performed using k-fold cross-validation, where the final accuracy is calculated as the average across all folds. The framework accepts as input an explicit graph representation constructed from four text files, each encoding different aspects of the graph data:

- A.txt: contains the graph adjacency matrix, represented as a list of index pairs (i, j) indicating the presence of an edge between node i and node j.
- graph_indicator.txt: specifies the association between nodes and graphs, indicating for each node which graph it belongs to.
- graph_labels.txt: provides the graph classification labels used for model training and evaluation.
- node_attributes.txt: contains node features, described as numerical vectors representing specific attributive properties of each node.

Our contribution focuses on the preprocessing phase, where we processed the raw gene expression, methylation, and CNA data like we have done in the preprocessing step for our models.
We conducted a series of experiments using the GCN and GAT models, employing a 10-fold cross-validation. The dataset, consisting of patient samples, was divided first into two distinct classes for binary classification and then into 4 classes. The results, in terms of accuracy and standard deviation, are reported in Table 3 and Table 4, respectively.

Table 3: Results [2 classes, 10-fold]

	1_agg_layer		2_agg_layer		3_agg_layer	
	Accuracy	std_dev	Accuracy	std_dev	Accuracy	std_dev
GCN	**62.311**	15.018	**72.953**	14.741	**72.953**	15.705
GAT	56.134	9.692	44.553	8.661	42.906	7.236

Table 4: Results [4 classes, 10-fold]

	1_agg_layer		2_agg_layer		3_agg_layer	
	Accuracy	std_dev	Accuracy	std_dev	Accuracy	std_dev
GCN	**45.446**	15.774	**38.053**	20.079	**39.955**	12.232
GAT	25.24	6.419	16.796	9.495	17.483	10.726

3.2 GGNN

The second method [17] we examine is a novel supervised deep learning approach, known as the Geometric Graph Neural Network (GGNN), which integrates geometric features, such as the invariant measure and the Ollivier-Ricci curvature, into deep learning to improve predictive power and interpretability. This method is designed specifically to predict cancer survival outcomes using multi-omics data.

The input genomic data were structured as a graph, integrating biological knowledge from PPI networks and pathway databases. The PPI network was sourced from the Human Protein Reference Database (HPRD), a manually curated database for reliable PPIs evidenced from published literature [18]. Key pathway data were obtained from the KEGG database, a widely used bioinformatics platform that offers detailed information on biological pathways and gene networks [19].

The prediction performance of the GGNN was compared with our model. For the GGNN model the data (gene expression, methylation and CNA) were structured with genes as rows and patients as columns, meaning that each entry (i, j) in the matrix represents the specific value for gene i in patient j. For each omic type, the genes used were selected by intersecting the gene list from the dataset for each patient with those available in HPRD and KEGG. We refined the data preprocessing step by performing the intersection of genes individually for each patient, rather than combining all genes between patients. This decision was based on the observation that different patients may have non-overlapping sets of genes present in their omics profiles. Consequently, the resulting data matrix included missing values for genes not present in a given patient, which we imputed by assigning a value of zero, interpreted as the absence of alteration or expression for that gene. Data were randomly divided into 60% for training, 20% for validation, and 20% for test. A C-index was computed for the validation and test sets, separately. We repeated the random splitting process 10 times and computed an average C-index for each experiment. This setup yielded a C-index of 0.63 for the test set.

To investigate the contribution of the sub-network of each omic type in modeling, we compared the predictive performance of individual omic types with the combined network, as shown in Table 5. We noticed that the combined network resulted in better predictive performance compared to single-omic data, meaning that the integration of multi-omics data enhances predictive power. RNA-Seq data showed better predictive performance than CNA and methylation.

Table 5: The C-index values for each omic type

RNA-net	CNA-net	Methyl-net	Combined-net
0.607	0.590	0.595	**0.629**

Table 6: Final C-index comparison

	Test
GGNN	0.629
Our Model	**0.737**

4 Data Analysis

Due to the initial division of the dataset, which was required in the ranking processes of the genes (Chapter 1.4), we were not able to implement the K-fold cross validation, hence we did not compute the variance between each possible fold in the dataset. On the other hand, we carefully divided the Train and Test set, so that they both represent the actual distribution of the entire dataset.

4.1 Classification Task

We can observe in table 2 that the model with better results across each class is "EdgeAttrGNN", it performs better across each possible class with respect to all other models.
Even though the average number of nodes per graph in the model "EdgeAttrGNN" is significantly lower than other models, it's able to outperform all the others; the number of nodes was reduced for this model because it required more capacity for the same amount of graphs in the dataset. We can also observe, for all the models, that the greater the complexity of the model and details for each graph, the fewer the number of average nodes per graph; this is due to the computational constraints available for this project.

The analysis of the first method from the literature [16] not only provides the accuracy of the models, but also gives information about the K-fold cross-validation and therefore the variance between them.
We can observe the results in Table 3 and Table 4.
The GCN model provides a higher variance prediction than the GAT model, which has lower accuracy, but also lower variance, across both tables and number of layers.

After analysis results for both our models and the other two methods from literature, we draw the following conclusions:
As expected for the classification task we found out that less class to classify helps the model differentiate better between user cases, and in general having more genes available for each graph gives more knowledge for the classification task.
What we didn't expect at first was the great difference in terms of performance between GCN and GAT layers, in which the first one is always better than the second one for both cases in which we consider edge attributes or not.

We also found out that for the GCN layer, providing a label for the edge improves its accuracy, but in the case of the GAT layer it seems that instead performance degenerated.
By comparing our results with the first method from the literature [16], the results demonstrate that our models achieve performance comparable to existing methods, and in many cases it exceeds.

4.2 Regression task

As observed in chapter 2.2 we have a great variability in the c-index metric, due to the way the batch is created, we have chosen to keep the training set batch always shuffled, to provide the model all the possible pair of user cases, and generalize better from them.
We supposed that, if we randomly shuffle the subset into pairs at each iteration, the average difficulty of distinguishing between the elements in the pairs may vary, being higher in some iterations and lower in others.

The analysis of the second method from the literature [17] provided distinguished results of individual omic types, we can observe them in Table 5.
As expected we gained better results with subnetwork of RNA data than all the other omics.
Also the combined network provides the best predictive performance than all the single omics.

As we can see in Table 6, our results are comparable to the state of art available, provided from the second method from the literature [17]; so our model provide a good alternative for this task.

4.3 Conclusion

In conclusion, based on the analysis made, we can say that our approach is solid and the work has been carried out effectively.
But we believe that with more data available in terms of number of user cases we could improve the model, at least for the classification task.

References

[1] STRING CONSORTIUM 2025. String ppi database. https://string-db.org, 2025.

[2] National Cancer Institute. mrna analysis pipeline. https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/Expression_mRNA_Pipeline/, 2025.

[3] Bo Li and Colin N. Dewey. Rsem: accurate transcript quantification from rna-seq data with or without a reference genome. *BMC Bioinformatics*, 12(1):323, 2011.

[4] Ana Conesa, Pedro Madrigal, Sonia Tarazona, David Gomez-Cabrero, Alejandra Cervera, Andrew McPherson, Michał Wojciech Szcześniak, Daniel J. Gaffney, Laura L. Elo, Xuegong Zhang, and Ali Mortazavi. A survey of best practices for rna-seq data analysis. *Genome Biology*, 17(1):13, 2016.

[5] Wikipedia. Documentation copynumbervariation. https://en.wikipedia.org/wiki/Longest_common_substring, 2025. Accessed: 2025-09-14.

[6] PyTorch Contributors. Adam optimizer reference. https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html, 2025.

[7] PyTorch Contributors. Reducelronplateau reference. https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html, 2025.

[8] PyTorch Contributors. Gcnconv reference. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GCNConv.html, 2025.

[9] PyTorch Contributors. Global mean pool reference. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.pool.global_mean_pool.html, 2025.

[10] PyTorch Contributors. Gatconv reference. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATConv.html, 2025.

[11] PyTorch Contributors. Nnconv reference. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.NNConv.html, 2025.

[12] PyTorch Contributors. Gatv2conv reference. https://pytorch-geometric.readthedocs.io/en/2.6.1/generated/torch_geometric.nn.conv.GATv2Conv.html, 2025.

[13] Mancinelli Francesco Colombari Martina, Lupo Davide Rosario. Testing results. https://drive.google.com/drive/folders/1ZwpJPiO6gMOK1x8cudAZQDpEEMxR9I-H?usp=share_link, 2025.

[14] Enrico Longato, Martina Vettoretti, and Barbara Di Camillo. A practical perspective on the concordance index for the evaluation and selection of prognostic time-to-event models. *Journal of Biomedical Informatics*, 108:103496, 2020.

[15] PyTorch Contributors. Mseloss reference. https://docs.pytorch.org/docs/stable/generated/torch.nn.MSELoss.html, 2025.

[16] qbxlvnf11. Repository graph neural networks for graph classification. `https://github.com/qbxlvnf11/graph-neural-networks-for-graph-classification.git`, 2020.

[17] Jiening Zhu, Jung Hun Oh, Anish K. Simhal, Rena Elkin, Larry Norton, Joseph O. Deasy, and Allen Tannenbaum. Geometric graph neural networks on multi-omics data to predict cancer survival outcomes. 2023.

[18] Renu Goel, H.C. Harsha, Akhilesh Pandey, and T.S. Keshava Prasad. Human protein reference database and human proteinpedia as resources for phosphoproteome analysis. *analysis, Mol. BioSyst.*, 2012.

[19] Minoru Kanehisa, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Kegg as a reference resource for gene and protein annotation. *Nucleic Acids Res.*, 2015.