Predicting novel miRNAs based on hairpin sequences: an ML approach

Savannah Linen[1*], Marc Silverman[1*], Chang Choi[1*]

[1]Project Group 1, Program in Bioinformatics, Georgia Institute of Technology, Atlanta, GA, USA

[*]Denotes equal contribution:

Author contributions: S.L., M.S., and C.C. conceived the project, designed the study, carried out model development and experiments, analyzed the data, and wrote the manuscript.

Figures available as PDFs at the end of the text following references. Source code for reproducibility: https://github.gatech.edu/slinen3/ML_proj

Abstract

MicroRNAs are RNA sequences that play an important role in gene expression across species. Generally containing 21 nucleotides, they act to regulate gene expression by aligning to the region that directs repression or cleavage of gene expression. Not only are they highly involved in gene expression, but they are also implicated in other pathologies such as cancer, viral infections, and heart disease. Precursors of these RNA molecules are hairpin sequences. Hairpins play a pivotal role in replication, making their identification an important problem in science. Machine learning (ML) has become important in this endeavor as such sequences are difficult to identify. To date, ML models have been successful in detecting homologous but not nonhomologous/species-specific miRNAs. Here, we develop two ML models in order to improve the detection of these sequences based on publicly available data. A support vector machine (SVM) and feed forward neural network were constructed, trained, and tested on three different datasets to expand upon the currently available models and improve the ability to accurately predict and identify hairpin sequences and miRNAs.

Introduction

MicroRNAs, or miRNAs, are a noncoding RNA consisting of 21 nucleotides discovered in *Caenorhabditis elegans* in the 1990s by *Lee et al* (Parveen et al., 2019; Stegmayer et al., 2019) that play a role in the posttranscriptional regulation of gene expression. They do so by aligning to the region directing translational repression or cleavage of a coding gene (Parveen et al., 2019). Most are transcribed into primary RNAs (pri-miRNAs), processed to precursor miRNAs (pre-miRNAs), and finally become miRNAs. Within a genome, miRNAs are found within the exons of non-coding genes, introns of coding and non-coding genes, and in intragenic regions (Ranganathan & Sivasankar, 2014). These highly conserved small RNA molecules are crucial to

gene expression and are implicated in other pathological processes including viral infections, cancers, neurological disorders, and heart disease (Parveen et al., 2019; Ranganathan & Sivasankar, 2014). Their regulatory nature gives importance to the further study of these molecules, specifically in the fields of drug discovery and oncology. Precursors of miRNAs (pre-miRNAs), or hairpins, are created in biogenesis and have distinct stem-loop structure. These hairpins play an essential role in replication initiation so predicting them aids in identifying and characterizing potential miRNAs in a genome (Bugnon et al., 2019). There are a large number of hairpin-like structures within a genome, making detection of pre-miRNAs difficult. To more efficiently identify pre-miRNAs, computational methods have become important and to date have been able to accurately detect homologous miRNAs but not nonhomologous or species-specific miRNAs (Stegmayer et al., 2019). Like previous studies, we will investigate multiple species datasets (Ben Or & Veksler-Lublinsky, 2021; Sheikh Hassani & Green, 2019). In this paper the drosophila melanogaster, Arabidopsis thaliana and c. elegans datasets were used for model development and testing (Stegmayer et al., 2019). This allows for a more comprehensive model to be developed as many studies to date only focus on single model organisms.

Currently, most published computational models use supervised learning to predict whether or not a sequence is a pre-miRNA or not. Unfortunately, accurately classifying de novo sequences remains a challenge. Unsupervised methods do exist as well but face the same challenge. Our goal in this paper is to develop a ML model to predict novel miRNAs accurately. To do this we will design two models and train them on a publicly available hairpin dataset (Bugnon et al., 2019). We will then perform rigorous analyses consisting of performance analyses, robustness testing, and generalizability. All code is available for reproducibility. Both the SVM and neural network designed here were able to predict outcomes across different datasets efficiently.

Methods

Data

The data used for this study was obtained from the publicly available dataset published in "Genome-wide hairpins datasets of animals and plants for novel miRNA prediction", by Leandro A. Bugnon, Cristian Yones, Diego H. Milone and Georgina Stegmayer (2019). Data consists of hairpin genome sequences from homo sapiens (hsa), arabidopsis thaliana (ath), anopheles gambiae (aga), caenorhabditis elegans (cel), and drosophila melanogaster (dme) and are divided into computed features and sequences. Computed features for predictions are in .csv format while sequences are in FASTA format. Each sequence is labeled as either positive or unlabelled, indicating whether it is a well-known pre-miRNA or that it does not yet have a known function. Each feature set contains 77 dimensions of information related to sequence, topology, and structure (Bugnon et al., 2019). More detail on data collection and structure was provided in the original study (Bugnon et al., 2019). Briefly, raw genomes were downloaded

from https://ncbi.nlm.nih.gov/genome, processed using HextractoR, predictions of the secondary structure of the sequences were performed with RNAfold, features were extracted using miRNAfe, and BLASTs were performed to match extracted sequences with known miRNAs in miRBase to label known pre-miRNAs (Bugnon et al., 2019). Python script for dataset statistics was provided, yielding feature importance, applying t-distributed Stochastic Neighbor Embedding (t-SNE) for visualization and producing histograms of the top three features with the highest average rank for each genome (Fig 1.).

Data Preprocessing & Splitting

We used both the hsa and cel datasets for our models. Due to the size of the hsa-unlabelled data, a representative subset of sequences were extracted and placed in a separate file. We begin by defining the subset size as 200,000 to ensure the size of the subset is comparable to the other datasets. The records within the FASTA file are then iterated through and random sequences are selected and appended to a list until the subset size is reached. The representative subset of sequences is then written to a new FASTA file with each sequence being assigned a unique identifier to maintain traceability.

To partition datasets into distinct training, validation, and testing datasets, the train_test_split function from sklearn.model_selection is employed. A split ratio of 60% training, 20% validation, and 20% test is used. Following data separating, labeled and unlabeled datasets are combined to generate a dataset containing both a positive and negative class for classification purposes. Randomization of sequences via shuffling is performed within each dataset to eliminate potential biases.

Support Vector Machine (SVM)

A supervised learning approach needs positive and negative data in order to create a binary classifier to discriminate between the two classes. In this case, we use real, known pre-miRNA as the positive class and pseudo/artificial non-re-miRNA as the negative class. In addition to having two distinct classes, a supervised learning model must have labeled data. Given labeled data containing two classes, the supervised algorithm creates a model that can predict which class a new point of an unknown class belongs to (Stegmayer et al., 2019). In this study, we used a support vector machine (SVM), a binary classifier, as one of our supervised algorithms. SVMs classify data using the kernel trick, mapping inputs into high dimensional feature spaces and separating classes by identifying the optimal hyperplane for separation that has a maximal margin between classes (Stegmayer et al., 2019). A linear machine defines the separation hyperplane. This is written as $y(x) = \omega^T \phi(x) + b$, where ω is the weight vector and b is bias. When searching for optimal parameters to maximize the margin between classes, the problem is a minimization of the Lagrangian equation $L(\omega, b, \alpha) = 1/2||\omega||^2 -$

$\Sigma^m{}_{i=1}\alpha_i\{y_i(\omega^T\phi(x) + b) - 1\}$, where α is a vector of Lagrange multipliers, with $\alpha_i \geq 0; \forall i$ (Stegmayer et al., 2019). This problem, a convex quadratic equation optimization problem, can be solved with the kernel $k_f(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ using Karush-Kauhn-Tucker conditions (Stegmayer et al., 2019). This model is generally applied with Gaussian kernels and default parameters (Stegmayer et al., 2019). The specific implementation of our classifier is described below.

Neural Network

Feed forward neural networks consist of a network of "neurons". The basic architecture is an input layer that moves to a variable number of hidden layers that result in an output layer. Each layer has multiple neurons, each of which obtain input from other neurons. A weighted sum is calculated by the neuron and an activation function is applied to the sum, generating the output for the following neurons in the model. Within this architecture the back-propagation algorithm works to train the model, updating synaptic weights via the backpropagation of a gradient vector that contains a collection of derivatives of error measures with respect to a certain parameter (Sazli, 2006). The following description can be found in more detail in Sazli, 2006. Here we give the essential equations to understand the model function. The error measure at each neuron, j, for the nth iteration can be written as:

$$e_j(n) = d_j - y_j(n) \quad (1)$$

where d_j is the desired output of the neuron (the correct class) and $y_j(n)$ is the actual output. For each iteration – each individual training instance – an instantaneous value can be calculated for the error energy:

$$\varepsilon_j(n) = \frac{1}{2}e^2{}_j(n) \quad (2)$$

This allows for the energy sum of all neurons to be calculated in visible output neurons via the summation of all error energy's for output layer neurons. The average squared energy can also be calculated for N patterns in a training set as follows:

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N}\varepsilon(n) \quad (3)$$

Both equations 2 and 3 are functions of bias levels and synaptic weights within the network, both of which can be manipulated within the back-propagation algorithm. Here we implement the model using batch mode. Batch mode acts by updating weights following each epoch. The process for updating weights begins with the output of the previous neuron. This can be written as:

$$y_j(n) = f(\sum_{i=0}^{m} w_{ji}(n)y_i(n)) \quad (4)$$

where m is the number of inputs without bias and f is a nonlinear activation function. Weight updates are proportional to the partial derivative of equation 2 with respect to the corresponding weight ($\partial\varepsilon(n)/\partial w_{ji}(n)$) and when the chain rule is applied the result is:

$$\frac{\partial\varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial\varepsilon(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial w_{ji}(n)} \quad (5)$$

Which can be further simplified using previously introduced equations to:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = -e_j(n)f'(\sum_{i=0}^{m} w_{ji}(n)y_i(n))y_j(n) \quad (6)$$

Finally, the correction, defined by the delta rule (Eqn. 7), is applied where η is the learning rate of the algorithm.

$$\Delta w_{ji} = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} \quad (7)$$

This is the underlying structure of the model we constructed; a simple feed forward neural network used for binary classification. Derivations and explanations of the underlying math are based on Sazli, 2006.

Experimental Implementation (Architecture)

SVM

For the SVM, we initially wanted to establish a working model. Unaware of the distribution of the 7 selected features determined from the Random Forest Classifier, we began by creating a simple model using only the cel (caenorhabditis elegans) data. To account for imbalances in the data, we sampled all of the positive labels, and an equal amount of randomly selected negative labels (Figure 2). The architecture consisted of two linear layers with rectified linear unit (ReLU) activation, a BCEwithLogitsLoss loss function, and an SDG optimizer. Input data and a hidden size of 64 were used as inputs. The model was run for 300 epochs with a learning rate of 0.01. After establishing the simple model, we wanted to build upon the complexity and determine the models generalizability across our datasets. To do this, we generated KDE plots for the 5 most important features (Figure 3). It is clear that the distribution of data across our 3 different species is similar for both positive and negative classes, so a generalizable model is plausible.

For the generalized SVM, we combined all the data across species, and uniformly distributed data is used for training as it was in the simple SVM model. We then increased the complexity of the model architecture. The GeneralizedSVM class takes in the input size, 2 hidden layer sizes, and dropout rate. The hidden layers contain 64 and 128 neurons respectively and dropout rate is set at 0.2. For the loss function, BCEWithLogits is once again used along with the torch SGD optimizer. The model runs for 40 epochs with a learning rate of 0.001. Within the model a method to clear the gradients of all optimized tensors is included to prevent poor convergence.

Neural Network

The neural network architecture consists of an input layer equal to the number of features in the dataset. Each dataset contained 78 features, of which we took the top 5 most important as input ('mfe', 'efe', 'dG', 'triplets0', and 'mfei4'). The importance of features was calculated using code provided with the data. Following the input layer there is a single hidden layer consisting of 64 neurons being activated with a Rectified Linear Unit (ReLU) activation function.

An output layer of 1 neuron activated using a sigmoid activation function that yields a probability score between 0 and 1 is the final layer. To compile the model, the binary cross-entropy (BCE) loss function is employed, measuring the difference between true and predicted labels output by the model. For loss minimization the Adam optimizer, an adaptive learning rate optimizer, is used. The model is trained using a learning rate of 1E-4 and batch size of 32 for 50 epochs. Due to severe data imbalance as evidenced in Figure 2, dataset balancing was performed. Balancing consisted of splitting positive and negative class labels, counting the number of positive labels (the underrepresented class), and randomly sampling an equal number of the negative data to match positive labels.

Results

SVM

Figure 4A displays changes in loss function for the training and validation data. The loss is not ideal, ending around 0.5, but it is worth noting the loss function used, BCEWithLogits, is a more complex loss function that calculates for both predicted probabilities and the actual target label. Even with relatively high loss, the accuracy of the model for true/false postive and negatives shows the model does a good job picking up on the true labels as shown in Figure 4B. For testing data, our 20% split was run 1 time. The total accuracy was 83.6%. The weighted confusion matrix displays the percent amongst all data within its respective true label (Figure 4B).

Neural Network

Experiments were conducted by reading in a single dataset at a time. Dataset details can be found in the *Data* section above. Preprocessing steps were then conducted as described above in *Data Preprocessing & Splitting.* Hyperparameters consisted of the following: learning rate = 1E-4, epochs = 50, batch size = 32, two dense layers of 64 and 1 neuron respectively using ReLU and sigmoid activation functions. The training procedure entailed loading in preprocessed data, balancing the data so the amount of negative and positive instances were equal, scaling features with StandardScaler(), splitting data with a test size of 0.2, calculating class weights using the compute_class_weight function from scikit-learn, and compiling the model.

Throughout the training process, training and validation loss and accuracy were displayed after each epoch for real-time analysis of model performance. Upon completion of the model, accuracy, precision, recall, F1-score, and ROC AUC were calculated. In addition to the metrics calculated, loss/accuracy curves and a confusion matrix were also generated as seen in Figure 4. These metrics were used to guide the adjustment of hidden layers, learning rate, and number of epochs. Following rigorous testing of various combinations of these variables, the most

successful model consisted of the aforementioned hyperparameters and yielded the metrics in Table 1.

Dataset	Test accuracy	Precision:	Recall:	F1-score:	ROC AUC
cel	0.84	0.89	0.73	0.80	0.91
ath	0.90	0.98	0.82	0.89	0.96
dme	0.92	1.0	0.83	0.91	0.96

Table 1. Performance metrics for each dataset when run on the neural network. All datasets were run with the same parameters.

To assess the generalizability of the model, the Friedman's Test was performed across data sets for each metric. The Friedman's Test is a nonparametric statistics analysis used to determine if values for each condition (dataset) differ significantly (Pereira et al., 2015; Rainio et al., 2024). The model was run 5 times on each dataset and the test was performed using the resulting metrics. With a p-value threshold of 0.05, there was no significant difference in any performance metrics across datasets, indicating the model generalizes well across datasets. Overall, the neural network model was able to effectively predict hairpin sequences within the data based on the top 5 features of the datasets.

Discussion and Conclusion

While our models display good accuracy and F1 scores given the limited time to create these models, one we encountered lied within the construct of the data we used. The data is classified as a '1' for a positive hit, but a '0' for a negative hit/unknown. In the original data, there is a large cluster of positive hits clearly located in a specific cluster in the dataset (Figure 1A), and other sparsely distributed positive hits throughout. This impairs the ability of users to decipher a negative hit from an unlabeled hit. In the authors' goal of locating new clusters of positive hits with different features than those typically found, the best options will be to extract additional true wet lab testing data, spliting the negative labels into 'true negative' and 'unknown'. Simple statistical models to determine if there are specific features outlier strands have in common can then be generated using true positives further away from the main cluster of true positives. We did not attempt this because we did not have the ability to validate our results based on the negatively labeled instances that could have contaminated the outcome. The outcome of these models show we have effectively applied our improved machine learning skills acquired through this course.

To improve our models, we would have liked to experiment more with the architecture and hyperparameters. For the SVM, the inclusion of a dynamic learning rate would be beneficial. Having a constant learning rate may not be the best choice as at different points in optimization, a higher or lower learning rate may be necessary. A more systematic method for choosing hyperparameters is necessary in future models as the majority of hyperparameters chosen were based on previous iterations of the model performance. In the future a surrogate model would be beneficial to address this issue (Qin et al., 2022). For the neural network, a more systematic method for tuning hyperparameters to fit each dataset best would be extremely beneficial. Despite the results of the model being generally well fit to each different dataset, performance could be improved if the model was optimized individually. To do this a gridsearch could be implemented, testing common hyperparameters including different learning rates, optimizers, and hidden layers. Further, the model was a supervised, simple feed forward neural network, meaning it was learning based off labeled data. Moving forward a more complex supervised neural network such as convolutional neural networks, recurrent neural networks, and long short-term memory models or unsupervised models including but not limited to density-based spatial clustering of applications with noise and anomaly detection algorithms should be investigated to better capture the complex nature of the data, specifically the imbalances and outliers of positive labels found within datasets. One hurdle that led to inefficient hyperparameter selection was processing unit limitations. We found that runn the models on Macs with M1 chips due to lack of access to GPU limited our ability to run training data, thus hindering the tuning of our hyperparameters.

Overall, we were able to successfully create two supervised learning models (an SVM and a simple feedforward neural network) to predict the presence of hairpin sequences in miRNA data for three different species. We found that not only that imbalanced data has a large impact on model efficacy and models will perform differently across different datasets, but also that the processing unit is extremely important. The generation of these models allowed us to experience the difficulties and nuances of building algorithms to work on real world data, which is often messy and inconsistent. Our ability to successfully implement two models on such inconsistent data displays our capabilities in the machine learning realm and has prepared us to move forward and apply such techniques in our future real-world endeavors.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (n.d.). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*.

Ben Or, G., & Veksler-Lublinsky, I. (2021). Comprehensive machine-learning-based analysis of microRNA–target interactions reveals variable transferability of interaction rules across species. *BMC Bioinformatics*, *22*(1), 264. https://doi.org/10.1186/s12859-021-04164-x

Bugnon, L. A., Yones, C., Raad, J., Milone, D. H., & Stegmayer, G. (2019). Genome-wide hairpins datasets of animals and plants for novel miRNA prediction. *Data in Brief*, *25*, 104209. https://doi.org/10.1016/j.dib.2019.104209

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/doi: 10.1109/MCSE.2007.55

McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Parveen, A., Mustafa, S. H., Yadav, P., & Kumar, A. (2019). Applications of Machine Learning in miRNA Discovery and Target Pre- diction. *Current Genomics*, *20*, 537–544.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library* (arXiv:1912.01703). arXiv. http://arxiv.org/abs/1912.01703

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., & Cournapeau, D. (n.d.). Scikit-learn: Machine Learning in Python. *MACHINE LEARNING IN PYTHON*.

Pereira, D. G., Afonso, A., & Medeiros, F. M. (2015). Overview of Friedman's Test and Post-hoc Analysis. *Communications in Statistics - Simulation and Computation*, *44*(10), 2636–2653. https://doi.org/10.1080/03610918.2014.931971

Qin, F., Yan, Z., Yang, P., Tang, S., & Huang, H. (2022). Deep-learning-based surrogate model for fast and accurate simulation in pipeline transport. *Frontiers in Energy Research*, *10*, 979168. https://doi.org/10.3389/fenrg.2022.979168

Rainio, O., Teuho, J., & Klén, R. (2024). Evaluation metrics and statistical tests for machine learning. *Scientific Reports*, *14*(1), 6086. https://doi.org/10.1038/s41598-024-56706-x

Ranganathan, K., & Sivasankar, V. (2014). MicroRNAs—Biology and clinical applications. *Journal of Oral and Maxillofacial Pathology*, *18*(2), 229. https://doi.org/10.4103/0973-029X.140762

Sazli, M. (2006). A Brief Review of Feed-Forward Neural Networks. *Commun. Fac. Sci. Univ. Ank. Series A2-A3*, *50(1)*, 11–17.

Sheikh Hassani, M., & Green, J. R. (2019). A semi-supervised machine learning framework for microRNA classification. *Human Genomics*, *13*(S1), 43. https://doi.org/10.1186/s40246-019-0221-7

Stegmayer, G., Di Persia, L. E., Rubiolo, M., Gerard, M., Pividori, M., Yones, C., Bugnon, L. A., Rodriguez, T., Raad, J., & Milone, D. H. (2019). Predicting novel microRNA: A

comprehensive comparison of machine learning approaches. *Briefings in Bioinformatics*, *20*(5), 1607–1620. https://doi.org/10.1093/bib/bby037

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Van Der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Waskom, M. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021

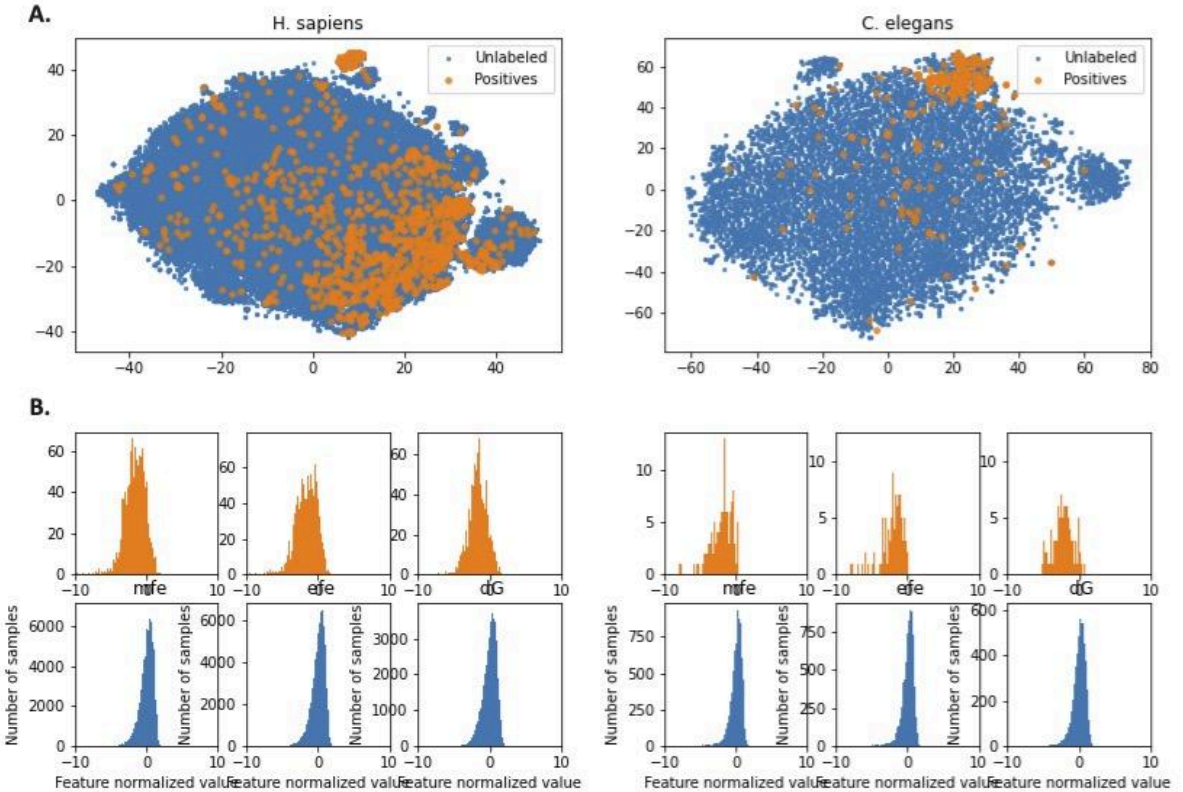
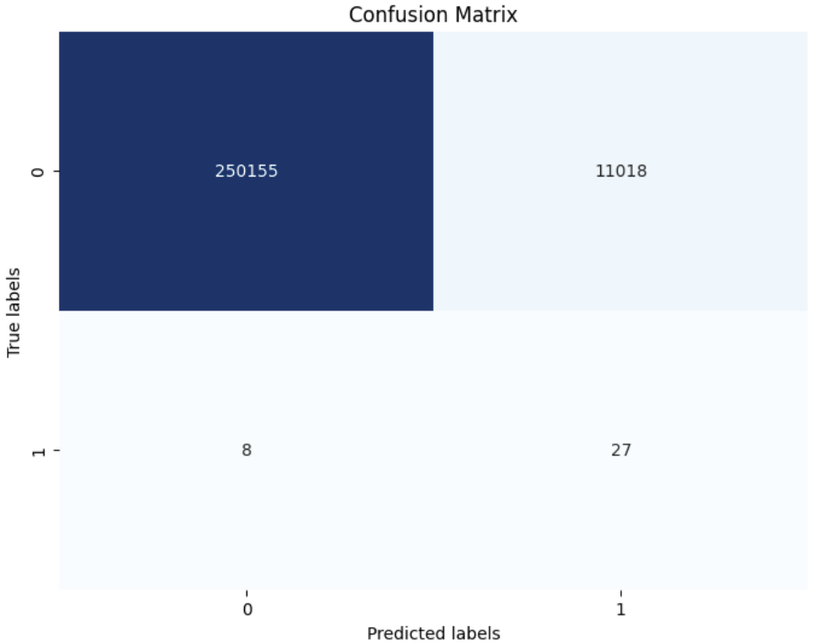


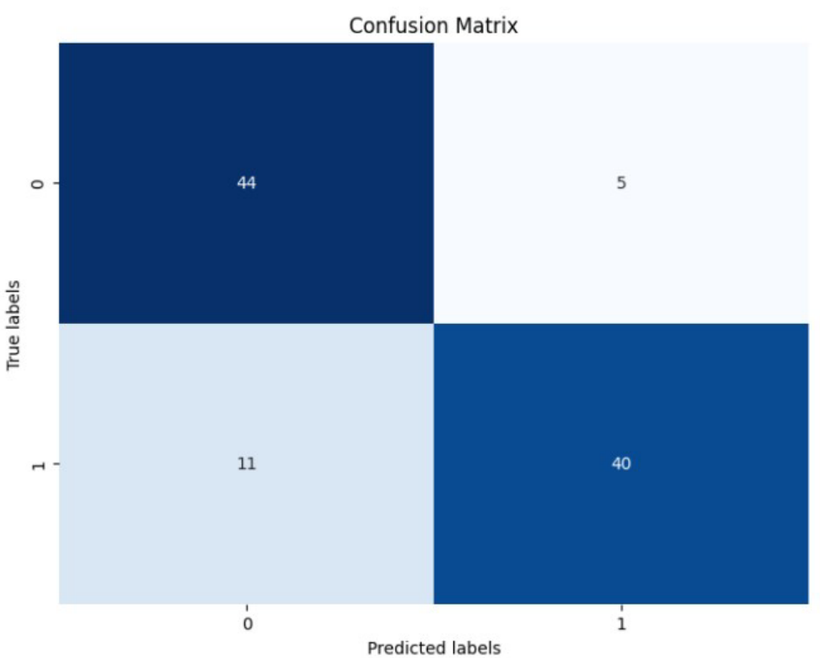
Figure 1. Data visualization provided by *Bugnon et al.* **A.** t-SNE projections of H. sapiens and C. elegans. Orange dots represent projections of well-known pre-miRNAs and blue represents a random set of unlabeled sequences. Sequences closer in the feature space are more similar. **B.** Histograms of top three features for H. sapiens (left) and C. elegans (right). mfe is minimum free energy, efe is normalized ensemble free energy, and dG is minimum fre energy normalized by length. Orange histograms represent the distribuion of these features in well-known miRNA while blue histograms represent other sequences.

A.

Un-balanced



Balanced



B.

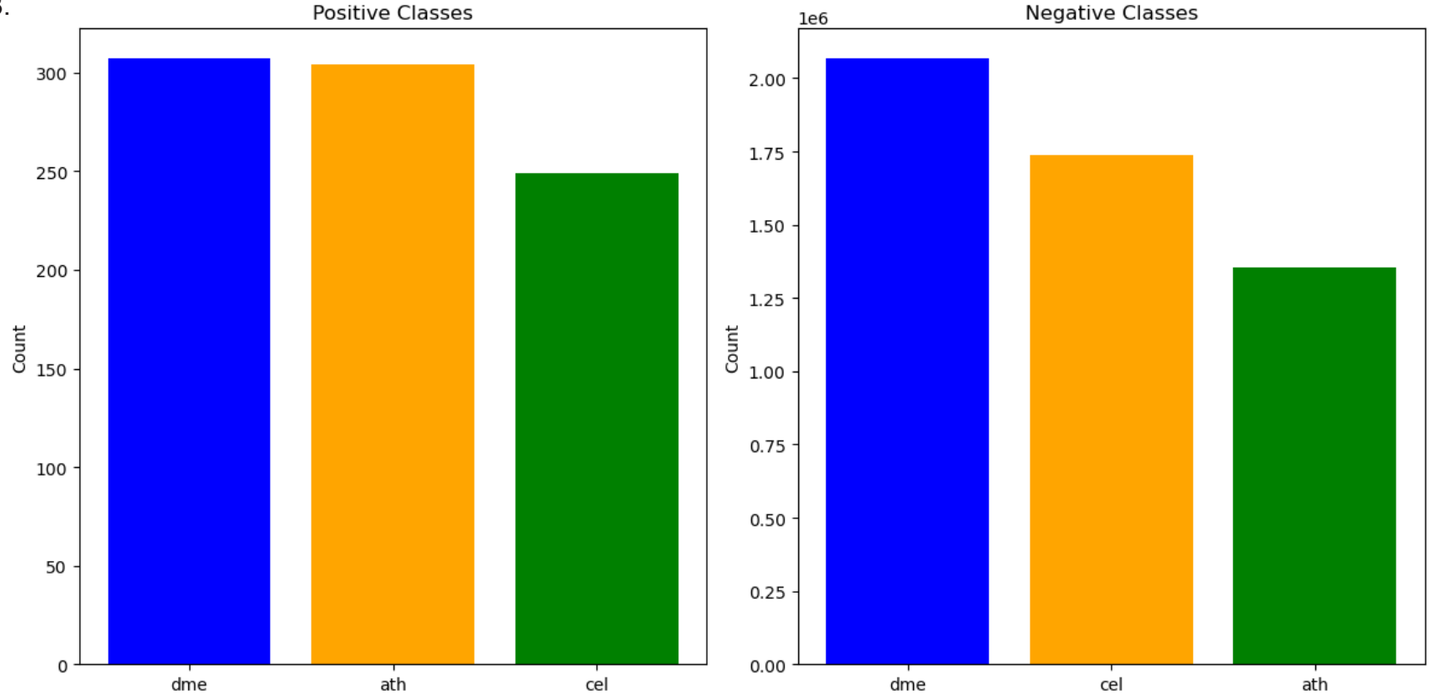


Figure 2. Visualization of data imbalance. A. Confusion matrices displaying data imbalance from the results of the cel data run on the neural network. **B.** Positive and negative class counts in each dataset (blue: dme, yellow: ath, green: cel).

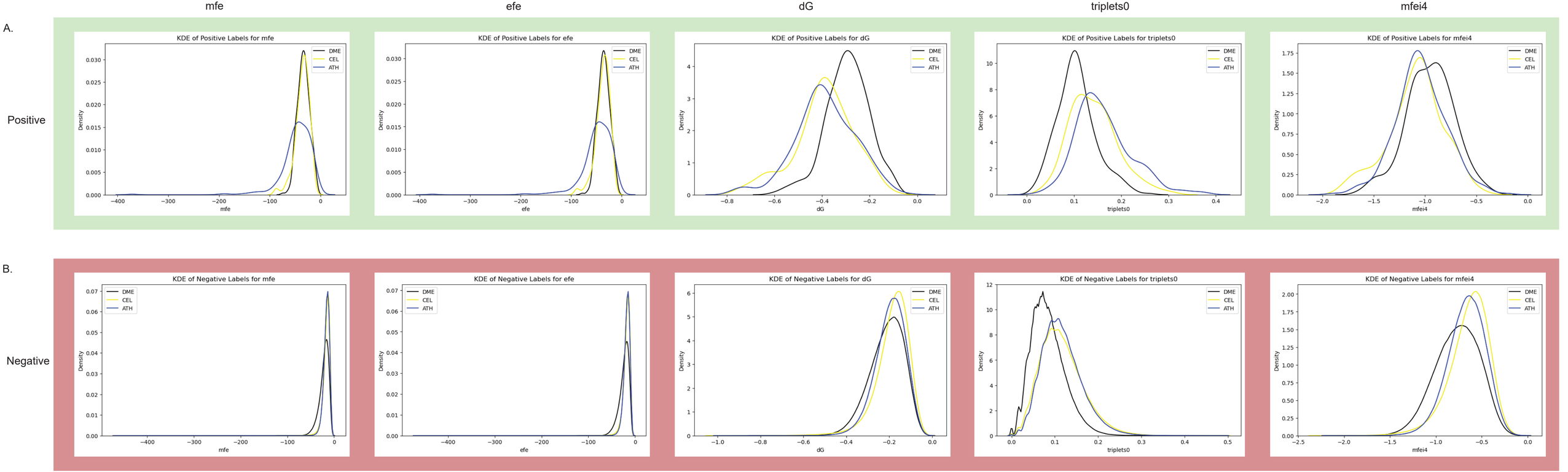


Figure 3. KDEs of extracted features based on feature importance. **A.** KDE plots for each of the top 5 features (mfe, efe, dG, triplets0, and mfei4) for positive labels. X-axes display the range of values for each feature. Yellow lines represent cel data. Blue lines represent ath data. Black lines represent dme data. density of Each graph is a selected feature positioned by importance from left to right. **B.** KDE plots for each of the top 5 features for negative labels. All other specifications the same as A.

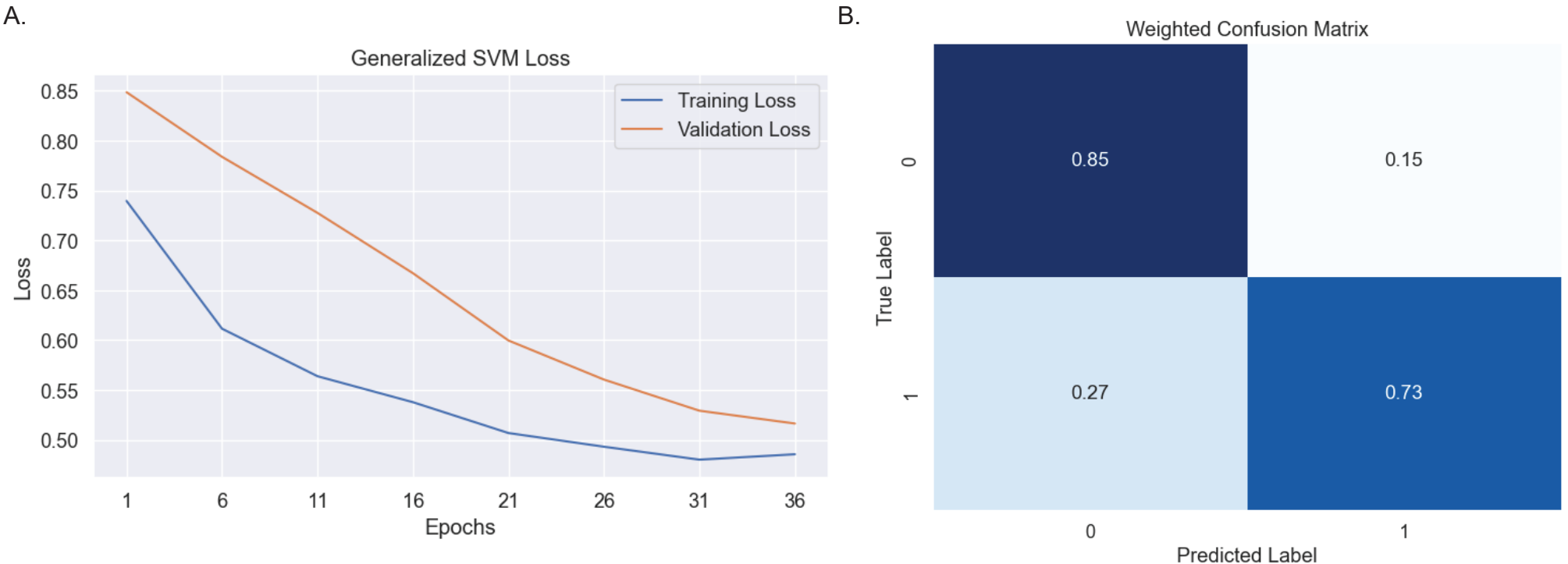


Figure 4. Loss curve and confusion matrix for SVM model. **A.** Loss curve for training (blue) and validation (orange) data. **B.** Confusion matrix for model performance.

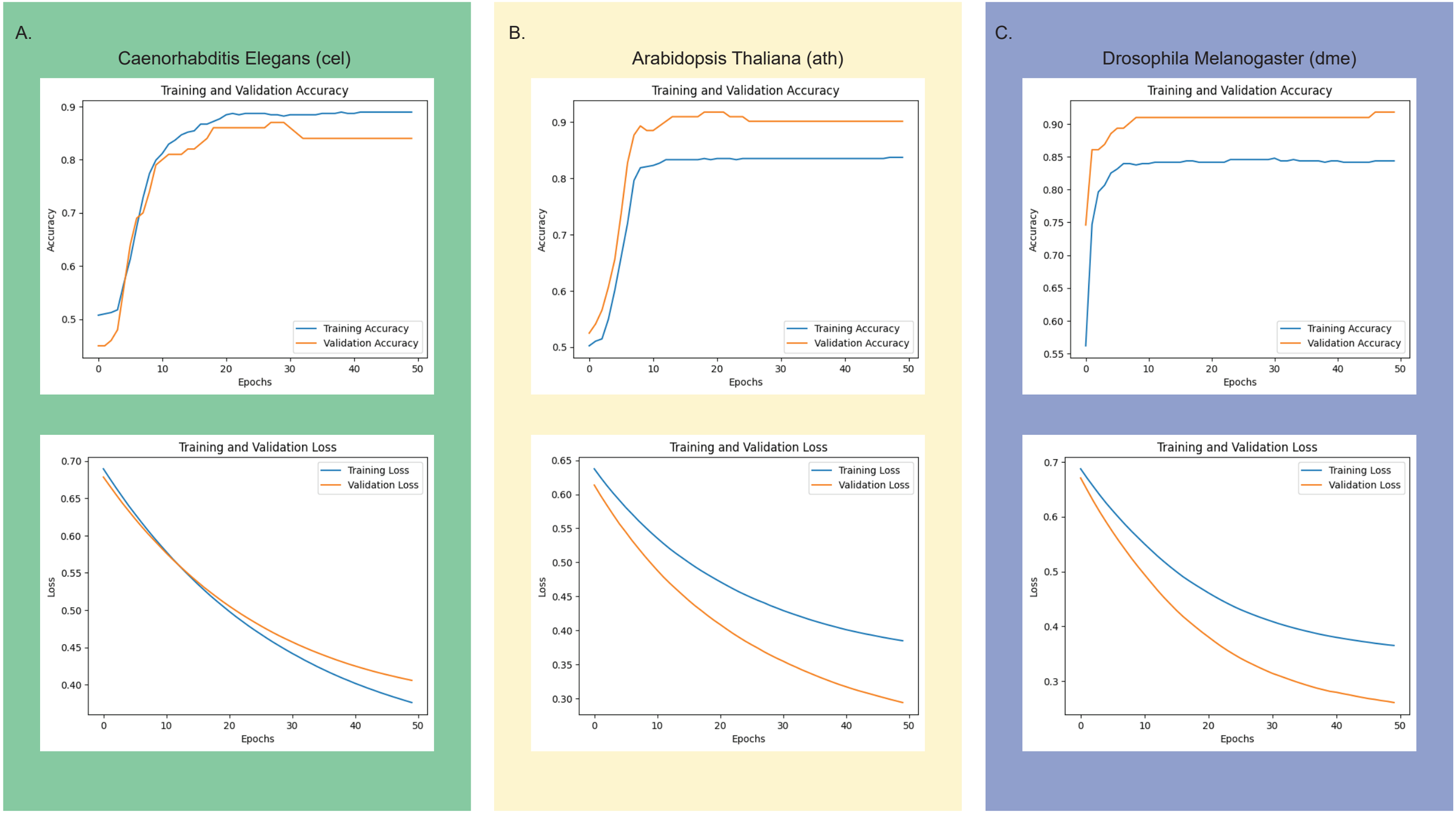


Figure 5. Training and validation loss and accuracy curves for three datasets tested on neural network model. **A.** Training and validation accuracy curve (top) and training validation and loss (bottom) over 50 epochs. Orange lines represent validation curves while blue lines represent training curves. **B.** Same as A. for ath data. **C.** Same as A. for dme data.

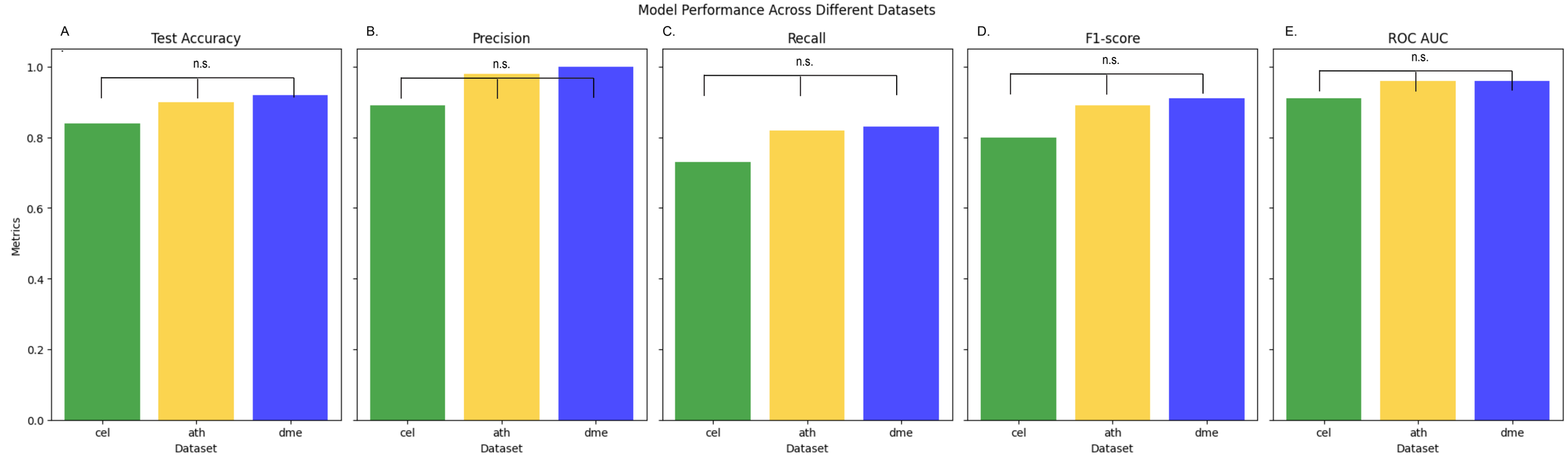


Figure 6. Performance metrics for neural network across datasets. **A.** Test accuracy for each dataset (cel, ath, dme). **B.** Precision for each dataset. **C.** Recall for each dataset. **D.** F1-score for each dataset. **E.** ROC AUC for each dataset.