CSI 5180. Topics in Artificial Intelligence: Machine Learning for Bioinformatics

Assignment 2

Marcel Turcotte

Version: Mar 5, 2025 18:07

🎯 Learning Objectives

- **Write** and **execute** a Jupyter Notebook.
- **Download** and **analyze** data in a cloud environment.
- **Prepare** biological data for a machine learning project.
- **Train** machine learning models.
- **Perform** hyperparameter tuning.

Assignment 2 builds upon our investigation into the identification of prokaryotic and eukaryotic promoters using machine learning techniques. Our objective is to delve into several key topics discussed during the course, such as class imbalance, hyperparameter tuning, and model selection. Additionally, we intend to implement deep neural networks to enhance our analysis. To this end, we will employ one-hot encoding techniques for our dataset and utilize convolutional neural networks (CNNs).

For those interested in extending their analysis, we propose a set of supplementary questions designed to offer additional insight and potential for extra credit. Specifically, we invite you to explore the application of Long Short-Term Memory (LSTM) networks or the integration of embeddings into your deep learning architectures. This exploration aims to assess whether these enhancements can further optimize the performance of your deep neural network models.

For a comprehensive overview of the promoter identification problem and a brief description of the dataset, please refer to Assignment 1.

📅 Submission

- **Deadline:**
  - The deadline for submitting your notebook is February 26 at 8 PM.
- **Individual Assignment:**
  - This task requires individual effort; collaboration is not permitted.
- **Submission Platform:**
  - Please upload your completed notebook to the Assignment section (Assignment 2) on Brightspace.

- **Submission Format:**
  - Please ensure that your submission comprises a Jupyter Notebook containing all results and an accompanying PDF version of the notebook. The PDF will facilitate discussions regarding the evaluation of your work with your teaching assistant.
  - The code should be executable on Google Colab. If your work involves additional libraries, include appropriate installation commands for those libraries.

Note for Submission: To avoid receiving a zero, ensure that your code executes correctly. You are responsible for verifying that your submission functions properly on a computer other than your own. Make sure all cells in your notebook are executable.

Deliverable

1. Jupyter Notebook

Develop a Jupyter Notebook that explicitly includes the course title, assignment details, and your personal information, such as your name and student identification number.

Just like for assignment 1, your notebook must read the data directly from the GitHub repository:

- CNNPromoterData on GitHub

To prevent errors when accessing data from GitHub, it is advisable to utilize URLs that direct to the raw data files. For example, the raw data for the file Arabidopsis_non_prom.fa can be accessed via the following URL. Here are the three links for your work.

- Arabidopsis_tata.fa (positive)
- Arabidopsis_non_tata.fa (positive)
- Arabidopsis_non_prom_big.fa (negative)

2. Data Encoding

To effectively utilize machine learning algorithms in bioinformatics, it is essential to transform biological sequence data into a format amenable to computational analysis. One prevalent method for achieving this is through the use of k-mers. In this context, each biological sequence is converted into a frequency vector that represents the occurrence of all possible nucleotide tuples of length k, where k is a user-defined parameter.

For a given sequence S and a specified k, the frequency of each k-mer, denoted as x_j, is calculated based on the tuple $[ACGT]^k(j)$ present in S. It is important to note that any tuple containing characters outside the nucleotide alphabet (A, C, G, T) are excluded from this analysis.

3. Datasets

Construct three datasets:

1. **positive** = 'Arabidopsis_tata.fa', **negative** = 'Arabidopsis_non_prom_big.fa'.
2. **positive** = 'Arabidopsis_non_tata.fa', **negative** = 'Arabidopsis_non_prom_big.fa'.
3. **positive** = 'Arabidopsis_tata.fa' + 'Arabidopsis_non_tata.fa', **negative** = 'Arabidopsis_non_prom_big.fa'.

The data is encoded using the FASTA format format, a widely recognized and straightforward file format in bioinformatics. A FASTA file comprises one or more sequences, each initiated by a single-line description that begins with a '>' character. This descriptor line, which can typically be disregarded during data processing, is followed by the sequence data itself. The sequence is usually distributed across multiple lines, each typically not exceeding 80 characters. This line-length convention enhances readability and was historically required for compatibility with older computer systems. In the dataset utilized for this assignment, the line-by-line convention is not implemented, as each sequence is presented on a single line.

In Assignment 1, we focused on two distinct datasets, each corresponding to a different type of promoter. The objective was to develop a separate model for each promoter type. In the current assignment, we introduce a third dataset in which the positive examples include sequences from both TATA-box-like and non-TATA-box-like promoters.

This expansion enables two primary lines of inquiry:

1. Evaluation of a unified machine learning model's capability to effectively classify both promoter types. While the task remains a binary classification of promoter versus non-promoter, the positive examples are now sourced from two different promoter categories.
2. Examination of the effects of class imbalance on model learning by utilizing datasets where the positive (minority) class constitutes 12%, 34%, and 39% of the total data, respectively.

4. Class Imbalance

To conduct our experiment, we will use logistic regression with default parameters and compare three distinct approaches to address class imbalance:

1. Baseline Experiment: No imbalance adjustment.
2. Class Weight Adjustment: Using the `class_weight` parameter set to `balanced` in scikit-learn's Logistic Regression, which adjusts weights inversely proportional to class frequencies.
3. Random Oversampling: Implemented with `RandomOverSampler` from imbalanced-learn.

The experimental protocol is as follows:

1. Partition the dataset by setting aside 20% as a hold-out set. Ensure that the `stratify` option is enabled to maintain the same proportion of positive and negative examples in both the training and hold-out sets.
2. Perform 10-fold stratified cross-validation on each of the three approaches (baseline, class weight adjustment, and random oversampling). For each, report the mean and standard deviation of balanced accuracy and the macro-averaged F1 score.
3. Train a model using each approach and evaluate it on the hold-out set.

The cross-validation process enables us to compute the average and standard deviation for each fold, providing insights into the variability of predictions across folds. The final assessment uses the entire training dataset, leveraging its larger size for more robust evaluation.

If computational resources are a constraint, consider applying 5-fold stratified cross-validation as an alternative to the 10-fold approach.

Write down your observations.

5. Best `k`

To streamline this assignment, we will focus exclusively on the third dataset, which includes positive examples from both TATA-box-like and non-TATA-box-like promoters. Based on your results from the previous section, please select the most effective strategy for addressing class imbalance.

Ideally, all experiments conducted in this assignment would be integrated into a single cross-validation framework. This approach would allow for the evaluation of potential combinations, such as random over-sampling with specific values of `k` and various hyperparameters, to determine the optimal configuration. However, to maintain simplicity, we will not implement this comprehensive procedure.

To identify the optimal value of `k` for k-mer encoding, employ cross-validation to evaluate values ranging from 2 to 6. For each value of `k`, calculate and report the mean and standard deviation of both the balanced accuracy and the macro-averaged F1 score.

Document your observations.

6. Hyperparameter optimization

Employing the previously utilized dataset, imbalance measure, and optimal `k` value, proceed with the following analysis:

Select two traditional machine learning algorithms, such as Logistic Regression and Random Forest. For each algorithm, identify key hyperparameters and conduct hyperparameter tuning as demonstrated in class. Compute and report both the mean and standard deviation for the balanced accuracy and the macro-averaged F1 score.

Determine which model and corresponding hyperparameter configuration yields the best performance. Record and discuss your observations.

7. One-hot encoding

K-mer encoding inherently results in the loss of some sequence information. In contrast, one-hot encoding is a prevalent method for representing categorical data. Utilizing our third *sequence* dataset, which comprises positive examples from both TATA-box-like and non-TATA-box-like promoter regions, generate a one-hot encoded representation of the data.

8. Convolutional neural networks (CNNs)

Convolutional neural networks (CNNs) are widely recognized as powerful deep learning models, particularly in bioinformatics. It is recommended to utilize Keras with TensorFlow serving as the backend due to its simplicity.

Typically, cross-validation experiments are conducted to optimize hyperparameters, such as the number of layers, filters, and their respective sizes, as well as activation functions, dropout and pooling strategies, optimizers, and regularization techniques. However, to streamline the assignment, we propose a predefined architecture that is known to perform well for the given problem. This sequential architecture is outlined as follows:

1. A one-dimensional convolutional (Conv1D) layer with 32 filters, a kernel size of 5, and a `relu` activation function.
2. A max pooling 1D layer with a pool size of 2.
3. A second Conv1D layer with 32 filters, a kernel size of 5, and a `relu` activation function.
4. A global max pooling 1D layer.
5. A dense layer with 64 units employing the `relu` activation function.
6. A dropout layer with a dropout probability of 0.5.
7. A dense layer with 1 unit using a sigmoid activation function.

For this architecture, utilize `binary_crossentropy` as the loss function and the `adam` optimizer, with a batch size of 32.

Begin by reserving 20% of the dataset for testing purposes. To determine the optimal number of training epochs, create a plot with the loss value on the y-axis and the number of epochs on the x-axis. The plot should display two curves: one for training loss and another for validation loss. Note that the validation set is automatically generated by the model's `fit` method using a `validation_split` of 10% (0.1) from the training data. The optimal number of epochs is identified as the point where the validation loss begins to increase, indicating potential overfitting.

Once the optimal number of epochs is established, proceed to train the model using this epoch count and evaluate its performance on the reserved test set.

Ensure comprehensive documentation of your observations throughout the process, detailing any insights and implications derived from the model's performance and behavior.

9. Hyperparameter Tuning for Convolutional Networks (Optional)

> **!** Important
>
> Our exploration represents only an initial foray into the potential applications of deep learning models with this dataset. Numerous additional models and experimental approaches could be investigated. Outlined below is a series of optional suggestions. While these are not required to achieve a perfect score of 10/10, each completed experiment may earn you an additional bonus point, up to a maximum of 2 points, representing a 20% increase in your total score.

Conduct a thorough experiment aimed at optimizing the hyperparameters of convolutional neural networks. Consider the following strategies:

- Adjust the number of convolutional layers to explore their impact on model performance.
- Modify the number of kernels per layer to assess how it influences feature extraction capabilities.

- Experiment with different filter sizes to determine their effect on capturing spatial hierarchies.
- Evaluate various activation functions to identify their role in non-linear transformations.
- Alter dropout probabilities to investigate their efficacy in mitigating overfitting.

Meticulously record your findings throughout the process, and present a detailed report on the optimal network architecture identified.

10. Long Short-Term Memory (LSTM) (optional)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to effectively capture temporal dependencies and long-range correlations in sequence data. However, Long Short-Term Memory also require more resources for training.

Just like for convolutional neural networks, cross-validation experiments would be conducted to optimize hyperparameters. However, to streamline the assignment, we propose a pre-defined architecture that is known to perform well for the given problem. This sequential architecture is outlined as follows:

- One LSTM layer with 64 units.
- One dense layer with 32 units and `relu` activation function.
- One Dropout layer with dropout probability 0.5.
- One dense layer with one unit and sigmoid activation function.

For this architecture, utilize `binary_crossentropy` as the loss function and the `adam` optimizer, with a batch size of 32.

Begin by reserving 20% of the dataset for testing purposes. To determine the optimal number of training epochs, create a plot with the loss value on the y-axis and the number of epochs on the x-axis. The plot should display two curves: one for training loss and another for validation loss. Note that the validation set is automatically generated by the model's `fit` method using a `validation_split` of 10% (0.1) from the training data. The optimal number of epochs is identified as the point where the validation loss begins to increase, indicating potential overfitting.

Once the optimal number of epochs is established, proceed to train the model using this epoch count and evaluate its performance on the reserved test set.

Ensure comprehensive documentation of your observations throughout the process, detailing any insights and implications derived from the model's performance and behavior.

11. Embedding (optional)

Embedding is a technique used to convert discrete data, such as words or sequences, into continuous vector spaces, facilitating their use in machine learning models. When applied to protomer DNA sequence recognition, embeddings transform these sequences into dense vectors that capture syntactic and semantic information. This enables models to identify patterns and relationships within the sequences more effectively, enhancing their ability to recognize and differentiate specific protomer sequences.

You can either train your own embedding or use a pre-trained embedding.

12. Cross-Species (optional)

We have demonstrated the capability to train a single model using positive examples originating from two distinct promoter categories: TATA-box-like and non-TATA-box-like. An intriguing avenue for further investigation would be to assess the cross-species applicability of such a model. Specifically, evaluating whether a model trained on data from species **a** can be effectively utilized to analyze corresponding data from species **b** could significantly enhance our ability to interpret genomic information from newly sequenced species.

Essentially, this means filling a table like this one where the row indicate the species used for training and the column the species used for testing.

	Arabidopsis	Mouse	Human	Bacilus	E. coli
Arabidopsis					
Mouse					
Human					
Bacilus					
E. coli					

Each cell needs to contain the balanced accuracy and macro-averaged F_1 score.

Evaluation Criteria

The report, which is your Jupyter Notebook, should comprehensively document the entire process followed during this assignment. The Jupyter Notebook must include the following:

- Your name, student number, and a report title.
- A section for each step, containing the relevant Python code and explanations or results.
  - For sections requiring Python code, include the code in a cell.
  - For sections requiring explanations or results, include these in a separate cell or in combination with code cells.
- Ensure logical separation of code into different cells. For example, the definition of a function should be in one cell and its execution in another. Avoid placing too much code in a single cell to maintain clarity and readability.
- The notebook you submit must include the results of the execution, complete with graphics, ensuring that the teaching assistant can grade the notebook without needing to execute the code.

Resources

If you do use AI assistance, thoroughly document your interactions. Include the tools and their versions in your report, along with a transcript of all interactions. Most AI assistants keep a record of your conversations. The recommended practice is to create a new conversation specifically for the presentation and consistently reuse this conversation throughout your work on the presentation. Ensure that this conversation is solely dedicated to the presentation. Submit this conversation transcript in the reference section of your summary.

Questions

- You may ask your questions in the Assignment topic of the discussion forum on Brightspace.
- Alternatively, you can email the teaching assistant. However, using the forum is strongly preferred, as it allows your fellow students to benefit from the questions and the corresponding answers provided by the teaching assistants.

References

Umarov, Ramzan Kh, and Victor V Solovyev. 2017. "Recognition of Prokaryotic and Eukaryotic Promoters Using Convolutional Deep Learning Neural Networks." Edited by Igor B Rogozin. *PLoS ONE* 12 (2): e0171410. https://doi.org/10.1371/journal.pone.0171410.