

Practical Lab (TP): Functional Annotation of Genomic Sequences Using Machine Learning

Nicolas Parisot and Sergio Peignier (based on a notebook from Thomas Nelson)

1 Introduction

Biology, and omics sciences in particular (genomics, transcriptomics, metagenomics, etc.), generate massive amounts of heterogeneous big data aimed at explaining complex cellular and molecular processes. To handle this avalanche of data, machine learning tools have become invaluable allies for extracting meaningful insights: predicting protein binding sites, classifying cancer types, predicting disease progression, precision medicine, genome-wide association studies, image classification, and microscope image denoising, among others.

The objective of this practical lab is to familiarize yourself with machine learning concepts, specifically focusing on how to evaluate the performance of a machine learning model. During this lab, we will apply machine learning to the functional annotation of genomic sequences. You will build, evaluate, and refine a model that predicts the biological function of genes based on their nucleotide sequences.

You are expected to build a Python Jupyter Notebook following the instructions below.

2 Available Data

You have access to a file named `human_data.fasta`, which contains 4,380 human gene nucleotide sequences. The biological function of each gene is known and indicated in the sequence description as an integer between 0 and 6.

The first step of your notebook is to import this information into two parallel, ordered lists:

- `list_sequences_index`: A list of strings where each element is the nucleotide sequence.
- `list_class_index`: A list of integers where each element is the corresponding class (function) of the sequence.

3 Information Retrieval and Feature Extraction

In our study, the underlying hypothesis is that a gene's nucleotide sequence determines its biological function. Therefore, we need a method to extract numerical information from these nucleotide sequences.

Text analysis is a major application domain of machine learning, and many numerical feature extraction methods from text have already been developed. The most intuitive method transforms text into numerical data by counting the occurrences of each word. However, standard machine learning algorithms typically require feature vectors of identical size, making direct application difficult when texts contain differing numbers of unique words. A standard solution is to use a vocabulary-based dictionary to count word frequencies across all texts, ensuring uniform vector sizes.

Because our input text consists of nucleotide sequences rather than natural language, a preliminary tokenization step is required:

1. Implement a function that splits a nucleotide sequence string into a list of words (k-mers / words of a given length).
2. Drawing inspiration from traditional text mining, what strategies can you propose to maximize the information extracted from our studied sequences?
3. Use the `CountVectorizer` function from the `scikit-learn` library to implement your chosen tokenization and counting solution.

4 Machine Learning Model Construction

Now that we have transformed our 4,380 human sequences into numerical feature vectors, we can use them to train a machine learning model.

Choose one supervised classification model available in `scikit-learn` (e.g., Random Forest, Support Vector Machines, Logistic Regression, or Naive Bayes). If you do not know which model to choose you can try `MultinomialNB`.

- Briefly explain how your chosen model works and why it is suitable for this classification task.
- Use the `train_test_split()` function from `scikit-learn` to divide your dataset into an 80% training set and a 20% validation set. Use the `random_state` option to ensure reproducibility.
- Train your chosen model on the training set using the `fit()` method.

5 Performance Evaluation

Our model has learned from 80% of the data. We will now apply it to the remaining 20% validation set to evaluate its performance using `model.predict(X_test)`.

Rather than relying on a single metric, compute and analyze the following four quality metrics using `scikit-learn` metrics module:

1. **Accuracy:** Explain what accuracy measures and discuss its limitations, especially in multi-class or imbalanced datasets.
2. **Precision:** Explain how precision is calculated and what a high precision score signifies for functional annotation.
3. **Recall (Sensitivity):** Explain what recall measures and why missing a true functional class could be problematic in biology.
4. **F1-Score:** Explain how the F1-score balances precision and recall into a single metric.

6 Application to Other Species

You have two additional files containing nucleotide sequences from other species: `chimp_data.fasta` (1,682 chimpanzee sequences) and `dog_data.fasta` (820 dog sequences).

Apply your human-trained model to these two external datasets **without retraining it**, compute their performance metrics, and discuss the results observed across species.

7 (Bonus) Advanced Embeddings with Deep Learning and Transformers

Classical bag-of-words approaches (like `CountVectorizer` using k-mers) ignore long-range dependencies and the complex grammar of genomic sequences. In recent years, Transformer-based language models trained on massive genomic corpora have revolutionized DNA sequence representation.

Examples of state-of-the-art DNA foundation models include:

- **DNABERT / DNABERT-2:** <https://github.com/jerryji1993/DNABERT>
- **The Nucleotide Transformer:** <https://github.com/instadeepai/nucleotide-transformer>

Advanced Exercise: Replace the traditional `CountVectorizer` embedding with embedding representations extracted from one of these DNA foundation models (or pre-trained transformer embeddings). Redo the classification exercise, compute the evaluation metrics (Accuracy, Precision, Recall, F1), and compare whether the transformer-based embeddings improve the functional annotation performance compared to simple k-mer counts.