DNA Promoter Classification Using Classical Machine Learning

1. Introduction

Promoter regions play a fundamental role in the regulation of gene expression by serving as binding sites for transcription machinery. Automatically distinguishing promoter from non-promoter sequences is therefore a key task in bioinformatics and computational genomics.

This project explores the use of **classical machine learning algorithms** to classify DNA sequences as promoters or non-promoters using the *UCI Molecular Biology (Promoter Gene Sequences)* dataset.

The objective is to build a complete, end-to-end pipeline—from raw text sequences to engineered features—and compare several widely used machine learning models based on their accuracy and class-specific precision, recall, and F1-score.

2. Data and Preprocessing

The dataset is loaded directly from the UCI repository into a DataFrame containing three attributes:

- **Class**: promoter (+) or non-promoter (−)
- **id**: sequence identifier
- **Sequence**: raw nucleotide string

Some sequences contain tab characters, which are removed before analysis. Each DNA string is then split into individual characters to obtain a list of nucleotides for every sample. A custom Python dictionary maps each sequence index to its list of nucleotides plus the corresponding class label.

This dictionary is converted into a DataFrame where **each column represents a nucleotide position**, and the final column corresponds to the class label. Transposing the DataFrame ensures that:

- Each **row** = one DNA sequence
- Each **column** = a specific nucleotide position
- The last column is explicitly renamed `Class`

Exploratory summary statistics confirm:

- All nucleotide positions take one of four values: **A, C, G, T**

- The dataset is slightly imbalanced but contains examples of both classes

3. Feature Engineering

Since nucleotide symbols are categorical, they are converted into machine-readable numerical features using **one-hot encoding**. For each sequence position, four binary indicator variables are created (e.g., `pos0_A`, `pos0_C`, etc.).

The target labels are also encoded as binary indicators (`Class_+`, `Class_-`).

This produces a **high-dimensional sparse feature matrix**, characterized by:

- A relatively small number of samples
- Hundreds of binary input features

Such a configuration highlights the importance of choosing algorithms robust to sparsity and potential overfitting.

4. Models and Training

The following scikit-learn models are trained and evaluated:

- **K-Nearest Neighbors (KNN)**
- **Multi-Layer Perceptron (MLP)**
- **Decision Tree (DT)**
- **AdaBoost** (with decision tree base estimators)
- **Gaussian Naive Bayes (GNB)**
- **Support Vector Machines (SVM)**: linear, RBF, and sigmoid kernels

The dataset is split into training and testing subsets using `train_test_split`.
For each classifier, predictions are evaluated using:

- **Accuracy**
- **Precision**, **Recall**, and **F1-score** for each class (positive and negative)

These metrics are stored in a comparative summary table.

5. Results

Model	Accuracy	Precision (−)	Recall (−)	F1 (−)	Precision (+)	Recall (+)	F1 (+)

Model	Accuracy	Precision (−)	Recall (−)	F1 (−)	Precision (+)	Recall (+)	F1 (+)
KNN	~0.68	~0.83	~0.45	~0.59	~0.63	~0.91	~0.74
MLP	~0.77	~0.88	~0.64	~0.74	~0.71	~0.91	~0.80
Decision Tree	~0.68	~0.83	~0.45	~0.59	~0.63	~0.91	~0.74
AdaBoost	~0.95	1.00	~0.91	~0.95	~0.92	1.00	~0.96
Gaussian NB	~0.95	~0.92	1.00	~0.96	1.00	~0.91	~0.95
SVM-Linear	~0.86	1.00	~0.73	~0.84	~0.79	1.00	~0.88
SVM-RBF	~0.91	1.00	~0.82	~0.90	~0.85	1.00	~0.92
SVM-Sigmoid	~0.91	1.00	~0.82	~0.90	~0.85	1.00	~0.92

Key Findings:

- **AdaBoost** and **Gaussian Naive Bayes** achieve the highest accuracy (~95%), with strong class-balanced performance.
- **SVMs with RBF and sigmoid kernels** also perform very well, outperforming simpler baseline models.
- **KNN** and single **Decision Trees** underperform, likely due to data sparsity and limited sample size.

6. Discussion

Several insights emerge from the results:

Why Gaussian Naive Bayes performs well

The assumption of feature independence aligns reasonably well with **one-hot encoded nucleotide positions**, making GNB an effective choice despite its simplicity.

Why AdaBoost excels

AdaBoost aggregates multiple weak learners (typically decision stumps), enabling it to:

- Capture non-linear interactions across positions
- Reduce overfitting
- Maintain robustness in high-dimensional spaces

Performance of SVMs

Non-linear kernels (RBF, sigmoid) model complex promoter boundaries effectively, while the linear SVM performs well but slightly worse.

Limitations of KNN and Decision Trees

These methods struggle with:

- High-dimensional feature spaces
- Sparse binary encodings
- Small datasets

7. Conclusion and Future Work

This project demonstrates that **classical machine learning techniques**, especially **Gaussian Naive Bayes** and **AdaBoost**, can effectively classify promoter sequences using simple one-hot encoded features.

The study provides a reproducible pipeline that includes:

- Raw text sequence processing
- Nucleotide-level feature engineering
- Multiple model training
- Comprehensive metric evaluation

Future Directions

Suggested extensions include:

- Hyperparameter tuning using **GridSearchCV**
- Cross-validation for more robust generalization estimates
- Additional metrics such as **ROC-AUC** and **PR curves**
- Exploring **Random Forests**, **Gradient Boosting**, or **LightGBM**
- Interpretability analyses to identify influential nucleotide positions
- Comparing classical ML with **CNN/RNN sequence models**