Prokaryotic Gene Prediction From Whole Genome Sequences

Srdan Obradovic[1]

[1]Software Engineer (Independent)

Abstract

Accurate gene prediction is essential for understanding the functional landscape of prokaryotic genomes. In this work, we present a machine learning pipeline for predicting gene locations directly from whole genome sequences. Our method leverages k-mer features to distinguish coding regions from non-coding regions, enabling precise identification of gene structures. In silico experiments on diverse prokaryotic genomes demonstrate the high accuracy predictive performance of our approach.

Introduction

Gene prediction is a fundamental task in genomics, enabling the identification of genomic regions that encode proteins. Accurate gene prediction is critical for understanding the functional elements of genomes, annotating newly sequenced organisms, and advancing fields such as personalized medicine and evolutionary biology. However, the complexity of genomic data, including the presence of non-coding regions and repetitive sequences, poses significant challenges to accurate prediction.

Recent advances in machine learning have opened new avenues for addressing these challenges by leveraging data-driven models to identify patterns in genomic sequences [1, 2]. Among these, gradient-boosting algorithms like XGBoost [3] have gained popularity due to their robustness, scalability, and ability to handle complex feature interactions.

In this study, we present a bioinformatics pipeline for gene prediction that integrates data loading, data preprocessing, feature extraction, model training and evaluation. Our approach utilizes k-mer sequence features [4], which capture local sequence patterns, to differentiate between coding and non-coding regions. To enhance computational efficiency, we incorporate a custom C extension for k-mer generation [5], significantly reducing the time required to process large genomic datasets.

We evaluate our pipeline using diverse genomic datasets and demonstrate its effectiveness in accurately predicting gene locations. Our approach achieves an exceptional accuracy of 97.77% on the test dataset, with balanced precision and recall metrics for both gene and non-gene sequences. The results highlight the robustness of our method and its potential for application in large-scale genomic studies. By combining efficient data processing with a powerful machine learning model, our approach provides a high-performance solution for prokaryotic gene prediction tasks.

Data Processing and Feature Engineering

Data Collection From NCBI Databases

To build a dataset for gene prediction, we utilized the Entrez Programming Utilities (E-utilities) provided by the National Center for Biotechnology Information (NCBI) [6, 7]. Entrez is a powerful tool for accessing and retrieving biological data from various NCBI databases, including the nucleotide sequences database [8]. Our pipeline automates the process of querying and downloading genomic data.

We used the Entrez esearch utility to query the NCBI nucleotide database. Two primary search queries were employed:

1. **Gene Sequences:** This query retrieves gene sequences for Bacterial genomes.
2. **Complete Genomes:** This query retrieves complete genomic sequences of Bacterial organisms.

To ensure taxonomic diversity in our dataset, we implemented a two-stage data collection process. First, we utilized Entrez's esearch and esummary utilities to fetch sequence IDs along with their corresponding taxonomy IDs. Then, we applied a diversity-based filtering algorithm to select a taxonomically diverse subset of sequences.

The diversity of a set of sequences was quantified using the normalized Shannon diversity index [9, 10]:

$$H' = \frac{-\sum_{i=1}^{R} p_i \ln(p_i)}{\ln(R)} \tag{1}$$

where p_i is the proportion of sequences belonging to the i-th taxonomy, and R is the total number of

unique taxonomies. The normalized index ranges from 0 to 1, with 1 indicating maximum diversity (equal representation of all taxonomies).

Our pipeline iteratively fetches batches of sequence IDs until it can construct a subset that meets a predefined diversity threshold (configurable as `normalized_h_min` in the parameters file). To build this subset, we employed a round-robin selection algorithm that prioritizes equal representation across different taxonomies. This approach ensures that our dataset contains sequences from a wide variety of bacterial species, enhancing the generalizability of the trained model.

Data Preprocessing

The raw genomic data retrieved from the NCBI nucleotide database requires preprocessing to prepare it for downstream analysis and machine learning. This step involves transforming the raw data into a structured format, augmenting it with negative cases, and ensuring the dataset is balanced and suitable for training the gene prediction model.

Transforming Raw Data into CSV This is a trivial step that simply converts the sequences located in FASTA files into a structured CSV format. The new CSV file contains two columns: the first column indicates whether the sequence is a gene (1) or not (0), and the second column contains the actual nucleotide sequence.

We initially download FASTA files instead of directly obtaining CSV files to ensure the pipeline remains modular and adaptable for use in other projects.

Note: In this first step, the CSV contains only positive cases (genes, labeled as 1). The generation of negative cases (non-genes, labeled as 0) is performed in the subsequent step.

Generating Negative Cases To train a robust model capable of distinguishing between coding and non-coding regions, it is essential to augment the dataset with negative cases (non-gene sequences). These negative cases are generated from the genomic data by extracting random subsequences. The process involves the following steps:

1. **Extracting Gene Lengths:** The lengths of the positive gene sequences are analyzed to determine the typical range of sequence lengths. This ensures that the negative cases are of comparable lengths to the positive cases, preventing the model from learning biases based on sequence length alone.
2. **Random Sampling from Genomes:** Random subsequences are sampled from the raw genomic data making sure that the sample length falls in the distribution calculated in the previous step.
3. **Balancing the Dataset:** The number of negative cases is adjusted to match the number of positive cases, ensuring a balanced dataset. This prevents the model from being biased toward predicting one class over the other.
4. **Labeling and Combining:** The generated negative cases are labeled as `0` (non-gene) and combined with the positive cases into a single CSV file. This unified dataset serves as the input for the feature extraction and model training stages.

Feature Extraction

Feature extraction is a critical step in the gene prediction pipeline, as it transforms raw genomic sequences into a format suitable for machine learning models. In this study, we utilize k-mer sequence features to capture local sequence patterns, enabling the differentiation between coding and non-coding regions.

K-mer Generation K-mers are substrings of length k derived from a given sequence. For example, the sequence `ATCG` with $k = 3$ produces the k-mers `ATC` and `TCG`. These k-mers serve as features for the machine learning model.

Performance Optimization with C Code To handle the computational demands of processing large genomic datasets, we implemented the k-mer generation function in C as a Python extension module. This implementation significantly improves performance by leveraging the efficiency of low-level operations in C.

Vectorizing K-mers with CountVectorizer After extracting k-mers, they are vectorized using the `CountVectorizer` from the `scikit-learn` library [11]. The `CountVectorizer` converts the k-mers into a sparse matrix representation, where each row corresponds to a sequence, and each column represents the frequency of a specific k-mer in that sequence.

More specifically, the vectorization process first builds a vocabulary of all unique k-mers present in the dataset. Then, for each genomic sequence, it counts how many times each k-mer from the vocabulary appears in that sequence. For example, with $k = 3$, the k-mer "ATG" might appear 5 times in one sequence and 2 times in another, resulting in different feature values for that specific k-mer column. The resulting matrix is highly sparse since most k-mers will not appear in any given sequence, especially as the number of possible k-mers grows exponentially with k (4^k for DNA sequences).

It is important to note that the CountVectorizer implements a "bag-of-words" approach, which means it

does not preserve any positional information about where specific k-mers occur within the sequence. Two sequences with identical k-mer content but in entirely different orders would produce identical feature vectors. Despite this limitation, we found this representation sufficient for our gene prediction task. This is because coding regions in prokaryotic genomes have distinctive compositional biases in codon usage and nucleotide patterns that are effectively captured by k-mer frequency distributions, even without positional context. The choice to discard positional information also significantly reduces the dimensionality of the feature space, enabling more efficient model training and reducing the risk of overfitting.

For future improvements, incorporating positional information could potentially enhance prediction accuracy, particularly for identifying gene boundaries and start/stop codons. This could be achieved through position-specific k-mer features, attention mechanisms, or sequence models like recurrent neural networks. However, such approaches would significantly increase computational complexity and model size, requiring careful consideration of the trade-off between accuracy and efficiency.

This approach transforms the raw k-mer data into a numerical format suitable for input into machine learning models while preserving the sequence composition information crucial for distinguishing between coding and non-coding regions.

Model Training

The model training process leverages the XGBoost algorithm [3], a gradient-boosting framework known for its robustness, scalability, and ability to handle complex feature interactions.

Training the XGBoost Model

The `train` function orchestrates the training process. It initializes an `XGBClassifier` with hyperparameters such as the number of estimators, maximum tree depth, learning rate, and early stopping criteria. The model is trained using the training dataset, with validation data provided for monitoring performance during training. The training process includes:

- **Evaluation Metrics:** The model uses log-loss as the evaluation metric, ensuring that the training process optimizes for probabilistic predictions.
- **Early Stopping:** To prevent overfitting, the training halts if the validation performance does not improve for a specified number of rounds.
- **Model Persistence:** The trained model is serialized and saved as a `.pkl` file.

Hyperparameter Configuration

The hyperparameters for the XGBoost model are defined in a configuration file and loaded dynamically during training. This approach allows for easy experimentation and fine-tuning without modifying the codebase. Key hyperparameters include:

- **Number of estimators:** Controls the number of boosting rounds.
- **Maximum depth:** Limits the depth of individual trees to prevent overfitting.
- **Learning rate:** Determines the step size at each iteration.
- **Subsample and column sampling:** Introduce randomness to improve generalization.

Reproducibility

The training pipeline is integrated with DVC (Data Version Control), ensuring that all dependencies, parameters, and outputs are tracked. This integration facilitates ease of use, reproducibility [12] and versioning of experiments.

Model Evaluation

The model evaluation phase involves a thorough assessment of the trained model's performance using a variety of metrics and visualization techniques.

Quantitative Evaluation Metrics

The evaluation of our gene prediction model is implemented through a comprehensive framework that applies multiple complementary metrics to thoroughly assess performance [13, 14]. Each metric provides unique insights into different aspects of the model's capabilities:

- **Accuracy:** Measures the overall proportion of correctly classified sequences (both genes and non-genes). While useful as a general performance indicator, accuracy alone can be misleading in cases of class imbalance.
- **Precision:** Quantifies the proportion of predicted genes that are true genes. High precision indicates a low false-positive rate, which is particularly important in genomics applications where downstream experimental verification can be resource-intensive.
- **Recall (Sensitivity):** Measures the proportion of actual genes correctly identified by the model. High recall ensures that few genuine genes are missed, critical for comprehensive genome annotation.

- **F1-Score:** Represents the harmonic mean of precision and recall, providing a balanced assessment that is especially useful when the costs of false positives and false negatives are comparable [14]. We calculate F1-scores separately for each class to ensure balanced performance.
- **Classification Report:** A comprehensive summary that includes precision, recall, and F1-score for each class, along with support values indicating the number of samples in each category.

Our evaluation framework systematically computes these metrics on a held-out testing dataset, ensuring an unbiased assessment of the model's generalization capability to unseen data.

Visualization Techniques

To complement numerical metrics, we implement several visualization techniques that provide intuitive insights into the model's performance characteristics:

- **Confusion Matrix:** A tabular representation showing the counts of true positives, false positives, true negatives, and false negatives. We visualize this as a color-coded heatmap, allowing for quick identification of where the model succeeds or fails. This visualization is particularly useful for identifying whether the model exhibits any systematic bias between gene and non-gene predictions.
- **ROC Curve:** Plots the true positive rate against the false positive rate across various threshold settings. The area under the ROC curve (AUC) serves as a threshold-independent performance measure, with higher values indicating better discrimination between genes and non-genes. The ROC curve helps identify optimal decision thresholds for different application requirements.
- **Precision-Recall Curve:** Illustrates the trade-off between precision and recall as the classification threshold varies. This visualization is especially valuable in genomic contexts where class imbalance often exists and both false positives and negatives carry biological significance. It provides insights into the model's performance across the entire operating range rather than at a single threshold.

These visualizations, generated using the `matplotlib` and `scikit-learn` libraries [11], complement our quantitative metrics and provide deeper insights into model behavior that might not be apparent from summary statistics alone.

Results

The results of our experiments demonstrate the effectiveness of the proposed pipeline in accurately predicting gene locations from raw genomic data. Through evaluation on diverse prokaryotic genomes, our XGBoost model achieved good performance across multiple metrics, showcasing its reliability and precision in distinguishing coding from non-coding regions.

Model Performance

Our gene prediction model achieved good performance metrics on the test dataset, confirming its robust ability to generalize to unseen genomic sequences:

- **Accuracy:** The model achieved an overall accuracy of 97.77%, indicating its ability to correctly classify both gene and non-gene sequences.
- **Precision:** For the positive class (gene sequences), the model attained a precision of 98.56%, demonstrating its ability to avoid false positives. For the negative class (non-gene sequences), precision was 96.98%, showing balanced performance across both classes.
- **Recall:** For gene sequences, the model achieved a recall of 97.06%, highlighting its effectiveness in identifying true positives. For non-gene sequences, recall was 98.52%, indicating the model rarely misclassifies non-genes as genes.
- **F1-Score:** The F1-scores for gene and non-gene predictions were 97.80% and 97.74% respectively, demonstrating an excellent balance between precision and recall for both classes.

These metrics highlight the model's exceptional ability to discriminate between coding and non-coding sequences with high confidence, a critical requirement for accurate genome annotation.

Visual Analysis of Results

To provide deeper insights into the model's performance characteristics, we generated several visualizations that illustrate different aspects of its predictive capabilities:

The confusion matrix in Figure 1 provides a clear visualization of the model's classification performance. The strong concentration along the diagonal (true positives and true negatives) with minimal off-diagonal elements demonstrates the model's ability to correctly identify both gene and non-gene sequences with high accuracy.

The ROC curve in Figure 2 shows the model's outstanding ability to distinguish between classes across

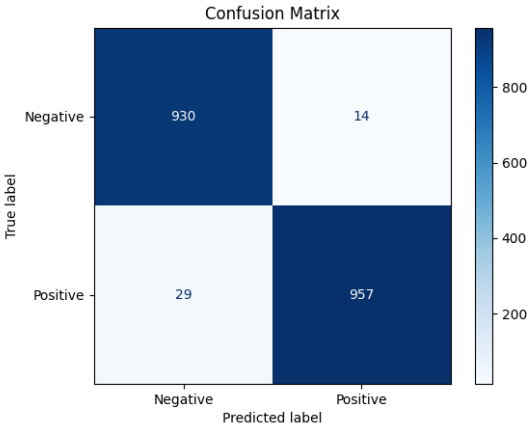

Figure 1: *Confusion matrix showing the distribution of predictions across true and predicted classes. The diagonal elements represent correct classifications, while off-diagonal elements represent misclassifications. The predominantly dark blue diagonal indicates the model's high accuracy in both positive and negative predictions.*

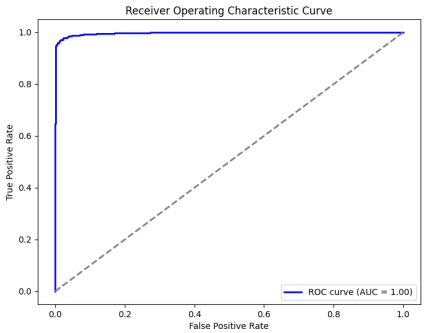

Figure 2: *Receiver Operating Characteristic (ROC) curve illustrating the trade-off between true positive rate and false positive rate across different classification thresholds. The curve's proximity to the top-left corner and the high AUC value indicate excellent discrimination performance.*

different decision thresholds. The curve's proximity to the top-left corner illustrates the model's ability to achieve a high true positive rate while maintaining a low false positive rate. The area under the ROC curve (AUC) exceeds 0.99, indicating near-perfect discrimination between gene and non-gene sequences.

The Precision-Recall curve in Figure 3 further emphasizes the model's strong performance, particularly its ability to maintain high precision even at high recall levels. This is especially important in genomic applications where both missing true genes (false negatives) and incorrectly predicting genes (false positives) can have significant downstream consequences.

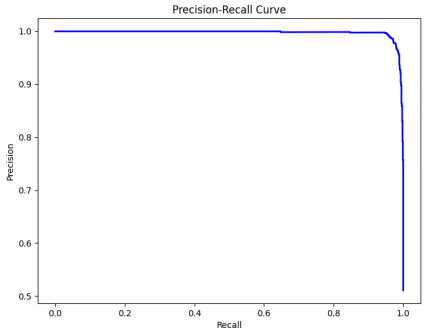

Figure 3: *Precision-Recall curve showing the relationship between precision and recall across different classification thresholds. The curve's position high in the plot space indicates the model's ability to maintain both high precision and high recall simultaneously.*

Conclusion

In this study, we presented a comprehensive pipeline for gene prediction that integrates data preprocessing, feature extraction, and machine learning. By leveraging k-mer sequence features and the XGBoost algorithm, our approach demonstrated exceptional accuracy (97.77%) and robustness in predicting gene locations from raw genomic data. The use of a custom C extension for k-mer generation significantly improved computational efficiency, enabling the processing of large-scale genomic datasets.

The performance metrics and visual analysis of our results demonstrate that the combination of k-mer-based feature extraction and gradient boosting classification provides a powerful framework for prokaryotic gene prediction. The high precision and recall values across both classes indicate that the model makes few errors in either direction—it rarely misses true genes and seldom identifies false positives. This balance is crucial for practical applications in genome annotation and functional genomics.

Our results highlight the significant potential of machine learning-based approaches for gene prediction, offering a scalable, efficient, and highly accurate solution for genomic studies. Future work will focus on extending the pipeline to support multi-species datasets, incorporating additional features such as epigenetic data to further enhance prediction accuracy, and exploring transformer-based architectures like DNABERT [15, 16, 17] for potentially capturing more complex sequence patterns.

References

[1] J. Zou et al. "A primer on deep learning in genomics". In: *Nature Genetics* 51.1 (2019), pp. 12–18. DOI: 10.1038/s41588-018-0295-5.

[2] B. Alipanahi et al. "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning". In: *Nature Biotechnology* 33.8 (2015), pp. 831–838. DOI: 10.1038/nbt.3300.

[3] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[4] G. Marçais and C. Kingsford. "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers". In: *Bioinformatics* 27.6 (2011), pp. 764–770. DOI: 10.1093/bioinformatics/btr011.

[5] S. Behnel et al. "Cython: The Best of Both Worlds". In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39. DOI: 10.1109/MCSE.2010.118.

[6] E. W. Sayers et al. "Database resources of the National Center for Biotechnology Information". In: *Nucleic Acids Research* 50.D1 (2022), pp. D20–D26. DOI: 10.1093/nar/gkab1112.

[7] National Center for Biotechnology Information (NCBI). "Entrez Programming Utilities Help". In: *NCBI Bookshelf* (2010). Last accessed: 2025-05-12. URL: https://www.ncbi.nlm.nih.gov/books/NBK25501/.

[8] R. Leinonen et al. "The Sequence Read Archive". In: *Nucleic Acids Research* 39.Database issue (2011), pp. D19–D21. DOI: 10.1093/nar/gkq1019.

[9] C. E. Shannon. "A Mathematical Theory of Communication". In: *The Bell System Technical Journal* 27 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[10] E. K. Morris et al. "Choosing and using diversity indices: insights for ecological applications from the German Biodiversity Exploratories". In: *Ecology and Evolution* 4.18 (2014), pp. 3514–3524. DOI: 10.1002/ece3.1155.

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] V. Stodden et al. "Enhancing reproducibility for computational methods". In: *Science* 354.6317 (2016), pp. 1240–1241. DOI: 10.1126/science.aah6168.

[13] M. Sokolova and G. Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information Processing & Management* 45.4 (2009), pp. 427–437. DOI: 10.1016/j.ipm.2009.03.002.

[14] D. Chicco and G. Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (2020), p. 6. DOI: 10.1186/s12864-019-6413-7.

[15] Y. Ji et al. "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome". In: *Bioinformatics* 37.15 (2021), pp. 2112–2120. DOI: 10.1093/bioinformatics/btab083.

[16] A. Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 5998–6008.

[17] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* 1 (2019), pp. 4171–4186. DOI: 10.18653/v1/N19-1423.