UNIVERSITY OF SALERNO

COMPUTER SCIENCE DEPARTMENT

Exploration of Gene Fusions through Combinatorial Approach and Machine Learning

Student

Eduardo Autore
Matr. 052250/1549

Teacher

Rocco Zaccagnino

Academic Year 2023-2024

Abstract

In the field of computational biology, the accurate association of RNA sequence reads (RNA-Seq) with their respective parent genes represents a fundamental goal. Addressing this challenge requires the application of advanced machine learning (ML) techniques. This study focuses on analyzing the effectiveness of representations based on **k-fingers**, exploring a hybrid between the context of **machine learning** and **combinatorial computation**, with a particular emphasis on the detection of gene fusions and the calculation of evaluation metrics. Studying the detection of gene fusions ultimately turns out to be the final objective and also the most interesting one since gene fusion is a mechanism of chromosomal rearrangement by which two (or more) genes unite into a single gene (**fusion gene**) and this is often associated with cancer. Chromosomal rearrangements that result in gene fusions are particularly widespread in sarcomas and hematopoietic neoplasms; they are also common in solid tumors. The splicing process can also give rise to more complex RNA patterns in cells. Gene fusions frequently affect tyrosine kinases, chromatin regulators, or transcription factors and can cause constitutive activation, enhancement of downstream signaling, and tumor development, as major drivers of oncogenesis. Knowing the importance of gene fusions in relation to cancer, we want to threshold the transcript datasets so as to obtain an excellent value (**threshold**) such as to allow us to obtain the greatest number of gene fusions possible with an excellent compromise in terms of metrics.

Summary

1 Introduction

Gene fusions or chromosomal rearrangements represent an important class of somatic alterations in cancer and can play significant roles in the early stages of tumorigenesis. The first chromosomal rearrangement associated with cancer was identified in 1960 as a translocation of chromosomes 9 and 22.

The main objectives of this analysis are to evaluate whether representations based on **k-fingers** can:

1. Highlight common regions between **RNA-Seq** reads and gene transcripts.
2. Train **ML models** to understand the complex mapping between reads and genes.
3. Use combinatorial calculations to control how the 2 genes are fused, obtaining evaluation metrics.
4. Evaluate the final metrics

To address this challenge, a three-phase approach has been adopted: initially, training **ML** models to assign **k-fingers** to specific genes, followed by the development of a specialized classifier that uses these **k-fingers** to associate reads with corresponding genes, and finally the use of various combinatorial calculation functions to compute metrics necessary for the evaluation of the *degree of gene fusion*.

Aware of the importance and implications associated with the discovery of a **gene fusion** compared to a single gene, we will simultaneously explore the field of gene fusions. Using **k-fingers** representations, we extend the methodology to identify *chimeric* reads, i.e., those assigned to two genes instead of one, and *non-chimeric* reads, i.e., those assigned to a single gene (without fusion). This alignment-free **machine learning** approach is supported by a rule-based classifier, leveraging knowledge acquired in the context of read-gene classification.

2 Factorization and Kfinger

In the analysis of sequences, factorization plays a fundamental role in understanding the structure and relationships within data sequences. The factorization of a sequence involves representing it as a sequence of substrings (factors) that, when combined, recreate the original sequence.

2.1 Preliminaries: Fingerprint e k-fingers

A factorization of a string s is a sequence of factors $F(s) = <f_1 f_2 ... f_n>$, where

$$s = f_1 f_2 ... f_n$$

The fingerprint of s with respect to $F(s)$ is the sequence of lengths of the factors, i.e., $L(s) = <|f1|, |f2|, ..., |fn|>$. The k-mers extracted from a fingerprint are called k-fingers.

2.2 Lyndon Factorization

Lyndon factorization is a particularly significant method of factorization as it ensures unique and useful properties for sequence analysis. A Lyndon word is defined as a sequence of characters that is lexicographically strictly smaller than each of its non-empty suffixes.

2.2.1 Direct Lyndon Factorization (CFL)

The direct Lyndon factorization of a sequence generates a sequence of Lyndon words $<f_1, f_2, ..., f_n>$, where each f_i is a Lyndon word. This process is essential for breaking down a sequence into fundamental elements that preserve Lyndon properties, providing a compact and unique representation of the sequence.

Example: CFL(AATTTCGAATTGCTTAACTC) = ⟨AATTTCG, AATTGCTT, AACTC⟩.

2.2.2 Inverse Lyndon Factorization (ICFL)

Inverse Lyndon factorization plays a crucial role when it is necessary to consider the reverse order of sequences. Producing the same sequence of Lyndon words but in reverse order, this factorization is of particular importance in the analysis of double-stranded biological sequences.

Example: ICFL(TACTATAGTTCTG) = ⟨TAC,TATAG,TTCTG⟩.

2.2.3 Importance of Lyndon Factorization

Lyndon factorization plays a crucial role in various contexts, thanks to its distinctive properties.

1. **Uniqueness and Linearity**

   Every word $w \in \Sigma^*$ has a unique factorization in terms of a Lyndon word $<f_1, f_2, ..., f_n>$, and this process can be performed in linear time with respect to the length of the word. Uniqueness and linearity provide an efficient and unambiguous representation of sequences.

2. **Preservation of Similarity Relations**

   Lyndon factorization preserves similarity relations between sequences. This means that similar sequences maintain a similar structure even in their Lyndon representation. Such preservation of relations is crucial for the detailed analysis of sequences, especially in biological contexts.

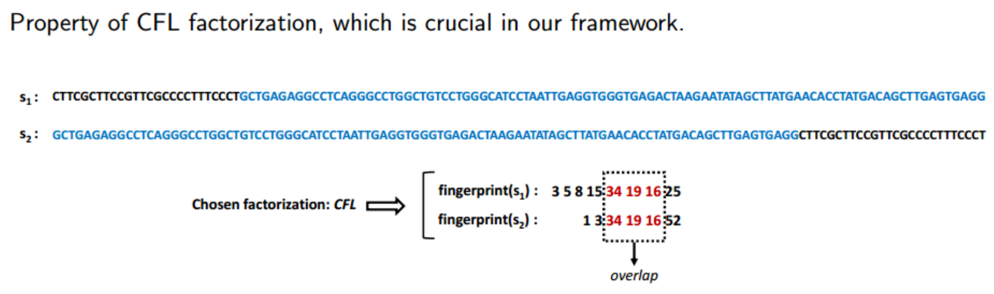

Property of CFL factorization, which is crucial in our framework.
s₁ : CTTCGCTTCCGTTCGCCCCTTTCCCTGCTGAGAGGCCTCAGGGCCTGGCTGTCCTGGGCATCCTAATTGAGGTGGGTGAGACTAAGAATATAGCTTATGAACACCTATGACAGCTTGAGTGAGG
s₂ : GCTGAGAGGCCTCAGGGCCTGGCTGTCCTGGGCATCCTAATTGAGGTGGGTGAGACTAAGAATATAGCTTATGAACACCTATGACAGCTTGAGTGAGGCTTCGCTTCCGTTCGCCCCTTTCCCT
Chosen factorization: CFL
fingerprint(s₁) : 3 5 8 15 34 19 16 25
fingerprint(s₂) : 1 3 34 19 16 52
overlap

Figure 1: Conservation properties.

2.3 Two Level Factorizations (CFL_ICFL)

The two-level factorizations, obtained through the combination of CFL (Direct Lyndon Factorization) and ICFL (Inverse Lyndon Factorization), represent an advanced approach in sequence analysis. This paragraph will explore the process of obtaining the factorization CFL_ICFL(w) from a word w and explain the reasons why this combination yields superior results compared to individual CFL and ICFL factorizations.

- **CFL ICFL(w) Factorization Procedure**

  1. **CFL(w)** $= \langle f_1, ..., f_t \rangle$: The sequence w is decomposed into factors through Direct Lyndon Factorization.

  2. **ICFL(f_i) for each** f_i: Apply Inverse Lyndon Factorization to each factor f_i obtained in the first phase, considering only those with a length $|f_i|$ greater than a threshold T.

- **Advantages Over CFL or ICFL Individually**

1. **Factorization Enrichment**: The union of CFL and ICFL enriches the factorization in terms of the number of factors, providing a more detailed view of the sequence's structure.

2. **Uniqueness and Linear Computation**: The combination of CFL and ICFL ensures the uniqueness of the factorization and the ability to perform computations in linear time with respect to the word's length, offering an efficient and accurate representation of sequences.

3. **Preservation of Relationships**: The conservation property, crucial in the detailed analysis of sequences, persists even in the union of CFL and ICFL.

2.4 Lyn2vec tool

Lyn2vec is an innovative methodology presented at prominent conferences such as AlCoB 2020 and 2021 (Algorithms for Computational Biology), aiming to address the challenge of the numerical representation of biological sequences. Its methodology is based on the factorization algorithm known as CFL, explained in the previous chapter, and in this context, we will explore the application of lyn2vec to a specific sequence s1.

Let's consider CFL as the factorization algorithm and set k, the length of the k-finger, to 3. We apply the CFL algorithm to the sequence s1:

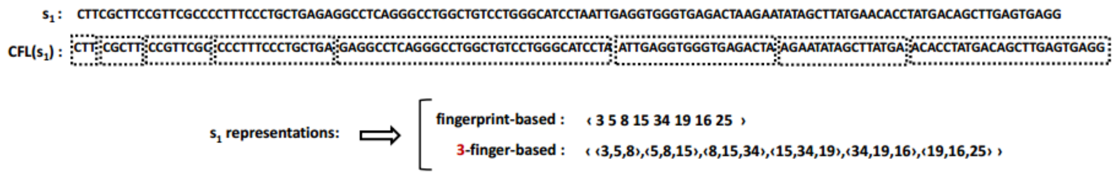

s_1 : CTTCGCTTCCGTTCGCCCCTTTCCCTGCTGAGAGGCCTCAGGGCCTGGCTGTCCTGGGCATCCTAATTGAGGTGGGTGAGACTAAGAATATAGCTTATGAACACCTATGACAGCTTGAGTGAGG
CFL(s_1) : CTT CGCTT CCGTTCGC CCCTTTCCCTGCTGA GAGGCCTCAGGGCCTGGCTGTCCTGGGCATCCTA ATTGAGGTGGGTGAGACTA AGAATATAGCTTATGA ACACCTATGACAGCTTGAGTGAGG
s_1 representations:
fingerprint-based : ‹ 3 5 8 15 34 19 16 25 ›
3-finger-based : ‹ ‹3,5,8›,‹5,8,15›,‹8,15,34›,‹15,34,19›,‹34,19,16›,‹19,16,25› ›

Figure 1: Lyn2Vec with K = 3.

To understand its functioning, let's introduce the concept of **superfingerprint** simply. Let's imagine having two ordered alphabets, like A, C, G, T and T, G, C, A, a string s, and a partitioning algorithm called F. The superfingerprint of s is obtained by **concatenating two fingerprints**: Lp(s), $, Lp'(s). Here, Lp(s) represents the fingerprint obtained by applying the algorithm F to the alphabet A, C, G, T on the string s, while Lp'(s) is the fingerprint obtained by applying the algorithm F to the alphabet T, G, C, A on the same string s. The symbol $ is used to mark the separation between the two fingerprints.

Lyn2vec takes as input a sequence read file in FASTA or FASTQ format and performs a factorization algorithm (specified as input) on each read to calculate its representation. In particular, lyn2vec produces the following types of representations:

1. The fingerprint.

2. The sequence of k-fingers extracted from the fingerprint of the read (given a specified k value as an input parameter).
3. The sequence of k-fingers extracted from the superfingerprint, after discarding the k-fingers containing the separator $.

Considering the two-level factorization (CFL_ICFL) explained in the previous chapter, let's illustrate with an example:

s = GCATCACCGCTCTACAG.

Using the CFL ICFL algorithm with a threshold **T = 30**, the factorization and fingerprint of s (with respect to the **"regular" ordering** A; C; G; T) will be respectively:

CFL_ICFL s = G, C, ATC, ACCGCTCT, ACAG e L(s) = 1, 1, 3, 8, 4 .

It is observed that the factorization with respect to the **reverse ordering** T; G; C; A produces the fingerprint 1, 2, 7, 7. Therefore, the superfingerprint will be given by:

S = 1, 1, 3, 8, 4, $, 1, 2, 7, 7

That is, by concatenating the two fingerprints interspersed with the separator $. Assuming k = 3, lyn2vec will produce one of the following three representations:

1. The fingerprint L(s) = 1, 1, 3, 8, 4 obtained from CFL ICFL s.
2. The sequence 1, 1, 3, 3, 8, 8, 4 of k-fingers extracted from L(s).
3. The sequence 1, 1, 3, 3, 8, 8, 4, 1, 2, 7, 7 of k-fingers extracted from S, after discarding those containing the separator $.

Final considerations

The resulting representations for sequence s1 demonstrate the effectiveness of lyn2vec in creating meaningful numerical representations for biological sequences. The sequence is transformed into fingerprints and 3-fingers, highlighting lyn2vec's ability to capture relevant information from the original sequence. Furthermore, the specific application of CFL and k = 3 provides an optimal configuration for capturing distinctive features of biological sequences, demonstrating the flexibility of lyn2vec in adapting to different contexts.

Lyn2vec has proven that the sequence of k-fingers is a suitable numerical representation for biological sequences that can be effectively learned by machine learning (ML) models. The use of this numerical representation opens new perspectives in understanding and analyzing biological sequences, providing a bridge between biological information and machine learning capabilities.

In the next chapter, we will discuss the use of this tool for extracting k-fingers from gene panel sequences to build a model capable of predicting and recognizing recurring patterns of k-fingers in sequences, thereby understanding to which gene they belong.

3 Choice and construction of the model

In this chapter, we will explore the Read-Gene classification problem using lyn2vec, which, as described in section 2.4, provides **three types of read representations**. Consequently, we addressed the Read-Gene classification problem in three different contexts, each considering a representation provided by lyn2vec. We organized our experiments into three groups, called tasks, addressing the following three questions:

- **T1**. ”How effective is the fingerprint produced by lyn2vec as a representation of a read in the Read-Gene classification problem?”
- **T2**. ”How effective is the sequence of k-fingers extracted from the fingerprints produced by lyn2vec as a representation of a read in the Read-Gene classification problem?”
- **T3**. ”How effective is the sequence of k-fingers extracted from the superfingerprint produced by lyn2vec as a representation of a read in the Read-Gene classification problem?”

Each task is composed of sub-tasks (or experiments), each dedicated to identifying and evaluating a classifier using a specific representation.

- **Task 1: Fingerprint-Based Classification**
  In the first task (T1), we explore the direct use of lyn2vec fingerprints as feature vectors for Read-Gene classification. Each fingerprint corresponds to a feature vector, and machine learning models are trained on these vectors to classify reads into their respective genes. This task provides insights into the effectiveness of lyn2vec fingerprints as standalone representations for error-free data.

- **Task 2: Sequence-Based Classification of k-fingers**
  Moving to the second task (T2), we investigate the use of k-finger

sequences, extracted from lyn2vec fingerprints, as feature vectors for Read-Gene classification. Here, we follow a two-phase process: (i) training models to assign each k-finger to a gene and (ii) using a specialized classifier (Section 4.1.2) that leverages the obtained result to assign each read to a class (or gene).

- **Task 3: Sequence-Based Classification of k-fingers from Superfingerprint**
  In the third task (T3), we extend the exploration to sequences of k-fingers extracted from lyn2vec superfingerprints. Similar to Task 2, this involves a two-phase classification process. Initially, models are trained to assign genes to the k-fingers extracted from superfingerprints, and subsequently, a specialized classifier is employed to assign reads to genes based on the assignments of k-fingers. This task aims to assess the effectiveness of superfingerprints in Read-Gene classification.

3.1 Data analyzed

For our experiments, we used the annotation (havana and ensembl_havana) of the 6040 human genes from chromosomes 1, 17, and 21, containing a total of 17,314 transcripts. Randomly, 100 genes were selected from them to obtain a small set of genes to evaluate the effectiveness of our embedding representation. For each of the 100 genes, 4 transcripts were randomly selected (for a total of **400 transcripts**). Subsequently, from each of these 400 transcripts, all 100-mers were extracted, resulting in a total of **797,407 substrings of length 100**.

3.2 Feature Extraction

The feature vector datasets were extracted from the collection of 797,407 input reads, each 100 bases long, in our dataset. Each fingerprint corresponds to a feature vector in task T1, and each k-finger extracted from a fingerprint (respectively, superfingerprint) corresponds to a feature vector in tasks T2 (respectively, T3).
In detail, a total of 10 factorization algorithms were considered: the four algorithms CFL, ICFL, CFL^d, and $ICFL^d$, plus the two algorithms CFL ICFL and CFL $ICFL^d$, applied for the three values 10, 20, 30 of the parameter T. For each algorithm, the fingerprint and superfingerprint of each input read were computed (using lyn2vec), resulting in a certain number of datasets for each experiment. Subsequently, the k-fingers for k ranging from 3 to 8 (a total of six values) were extracted from the datasets, obtaining k-finger datasets. Since the length of a feature vector must be constant and the number of elements (integer values) composing the feature vectors (fingerprints or k-fingers) is clearly variable, it is necessary to perform padding with trailing 1s to achieve a constant size across all feature vectors. Note that a k-finger can also be padded when the fingerprint is shorter than k.

3.3 Best classifier evaluation

Basic Methodology: For each experiment, the following steps were executed to produce a classifier:

Labeling: Each feature vector (fingerprint or k-finger) in the dataset was labeled to create a link to its class (i.e., the originating gene).

Learning: The following machine learning models were considered: **Random Forest (RF)**, **Logistic Regression (LR)**, and **Multinomial Naive Bayes (MNB)**. Each model was trained using labeled samples. Subsequently, the k-fold cross-validation technique (i.e., finding optimal hyperparameter values that produce satisfactory generalization performance) was used in combination with the GridSearchCV method to select the best model and obtain a classifier. The dataset was first split into two subsets using stratification: a training set (80% of the samples) and a test set (20% of the samples). Stratification ensures that all 100 classes (genes) considered are represented in both sets, maintaining the same proportions as the original dataset. Additionally, to minimize data loss, a common machine learning practice was employed, consisting of applying normalization within the cross-validation folds separately. In particular, we normalized the data using the MinMaxScaler technique. Normalization is used to map feature values (integer values in our case) to a fixed range (usually [0, 1]), providing better results than when features have values in different ranges. Next, k-fold cross-validation was performed on the training set (different k values were tried, and 5 gave good results), and at each k step (at each step, k - 1 folds were used to train the models, and one fold, called the test fold, was used for performance evaluation), an exhaustive search with specified parameters (GridSearchCV method) was applied, and for each combination of these values, the average performance of the considered classifier achieved on the current independent test fold was calculated. Finally, after discovering satisfactory hyperparameter values, retraining of the best model (the model showing the best performance) on the training set was performed, thus producing a classifier.

Evaluation: The generalization ability of the obtained classifier, i.e., the ability to achieve high performance on unseen samples, was evaluated by applying it to classify elements in the test set using the best parameters found in the previous step.

Best Model

From various conducted tests, based on the metrics, the best model has been identified as the one built upon **Random Forest (RF)**. Specifically, the RF model trained on the k-fingers extracted from the superfingerprints, computed with the CFL_ICFL^d algorithm and with T = 30, considering k = 8, achieved a weighted average precision of 0.91, recall of 0.77, and F1 score of 0.82 (evaluation phase).

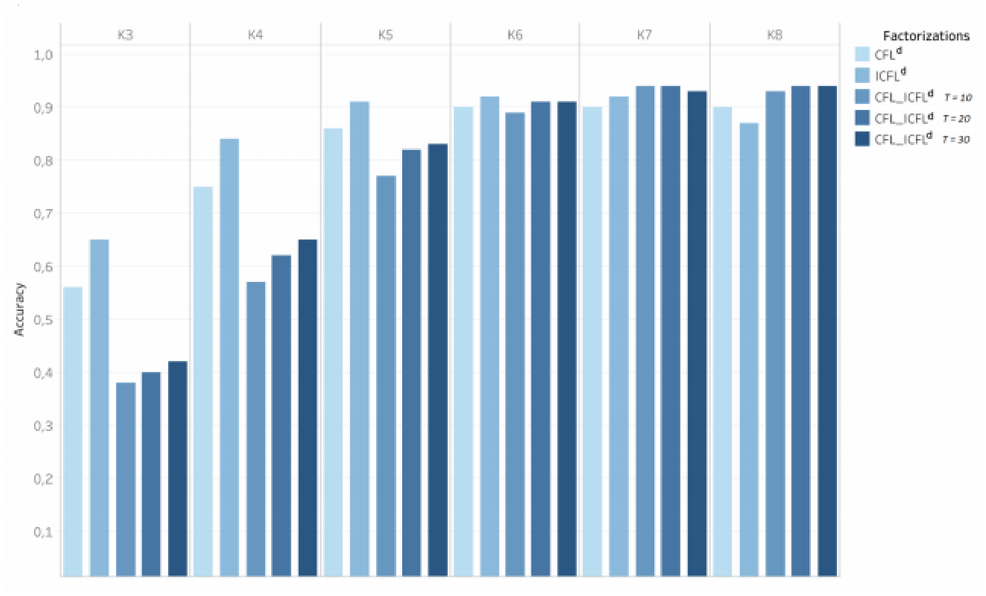


Figure 1: Accuracy reached with RF for each type of double-stranded factorization and k from 3 to 8.

After constructing the model, in the next chapter, we will explore the development of a read-gene classification system that takes the predicted k-fingers for each read and performs combinatorial calculations to compute a coverage percentage for each gene. This aims to determine whether the read is chimeric or non-chimeric.

4 Chimeric read-gene classification

Problem: The problem of chimeric read classification involves identifying genetic reads (reads, and specifically nucleotide sequences) that align with two genes instead of just one.
General Methodology: The methodology developed for read classification is generalized to implement a hybrid machine learning (ML) approach with a combinatorial approach. The k-finger classifier, discussed in Chapter 3, previously trained for gene read classification, is used. Subsequently, a rule-based classifier is defined to address the problem of chimeric gene read classification.

1. **Training the K-finger Classifier**: To train the k-finger classifier, a set of genes (gene panel) and their annotated transcripts are considered. For each transcript, all substrings of size equal to the specified window are extracted. Subsequently, k-fingers are extracted using the CFL_ICFL^d algorithm with specific parameters. The training dataset consists of all k-fingers labeled with the original gene.

2. **Adaptation of Factorization for Long Reads**: To compute k-fingers for long reads, an **adaptive factorization** guided by the factor set calculated in the training phase is developed. Each word is divided into substrings so that there exists an L-target-factor in the factor set that is a prefix of each substring.

3. **Classification of Long Reads into Chimeric/Non-chimeric**: Given the substrings obtained from adaptive factorization $w = w_1 ... w_n$, CFL_$ICFL^d$ is applied for each w_i (T = 30, k = 8). For each substring w_i, all k-fingers are **extracted**, resulting in an ordered list $<K_1, K_2, ..., K_q>$. The k-finger classifier is applied to this list, generating an **ordered list of genes assigned to the k-fingers**:

$$ListG(w) = <g_1, g_2, ..., g_q>$$

Based on this list, the repetitiveness of each gene is evaluated.

4. **Calculation of Gene Coverage Measure and Fusion Score**:

   The maximum repetitiveness of a gene is used to calculate a fusion score between two genes, indicating how likely the two genes are fused in a long read.

   The repetitiveness of a gene g in ListG(w) is defined as the number of pairs of consecutive elements in ListG(w) that are equal to g.

4.1 Calculation of coverage and fusion score

The gene coverage for gene g, indicated as **g-coverage**, is represented by the maximum sublist $<g_i, g_{i+1}, ..., g_j>$ in ListG(w), satisfying the following conditions:

- $g_i = g_j = g$,
- g is the gene with the maximum repetitiveness in ListG(w).

In various combinatorial algorithms used to assess the **degree of fusion** for different gene pairs, there is a common portion of code related to the fusion score calculation. The objective is to answer the question: Can we measure how likely genes g1 and g2 are to fuse in w?
To answer this question, the fusion score for two genes g1 and g2 in a long read w has been defined as follows:

$$score_w(g1, g2) = \frac{(P1 + P2 - P\cap)}{|ListG(w)|}$$

Where:

- **P1 (P2)**: represents the length of the g1-coverage (g2-coverage) list.
- $\boldsymbol{P\cap}$: is the number of common elements between the g1-coverage and g2-coverage lists.
- $\boldsymbol{|ListG(w)|}$: is the total length of the ListG(w) list.

Procedure for Each Gene Panel

1. Generate a chimeric and a non-chimeric dataset from the gene panel.
2. Calculate an **adaptive threshold** for the two datasets (chimeric and non-chimeric) based on the obtained metrics and, therefore, based on the number of *true positives*, *true negatives*, *false positives*, and *false negatives*, summarized with the **F1-score**. In the end, obtain a threshold that leads to a good compromise between various metrics, optimizing the number of returned chimeric reads.
3. For each input long read w (in both datasets) and for each pair of genes g1 and g2, calculate the g1 coverage and g2 coverage with respect to ListG(w).
4. If g1 coverage is not enclosed within g2 coverage (or vice versa), the fusion score (g1, g2) is then calculated.
5. Among all input gene pairs, select the pair that yields the highest score and label it as chimeric if that score is greater than or equal to the previously calculated adaptive threshold.

It is important to note that the $score_w(g1, g2)$ is normalized and varies within the range [0, 1]:

- **value 1**: indicates a perfect coverage of the ListG(w) list by the two genes.
- **value 0**: indicates the absence of coverage by the two genes.

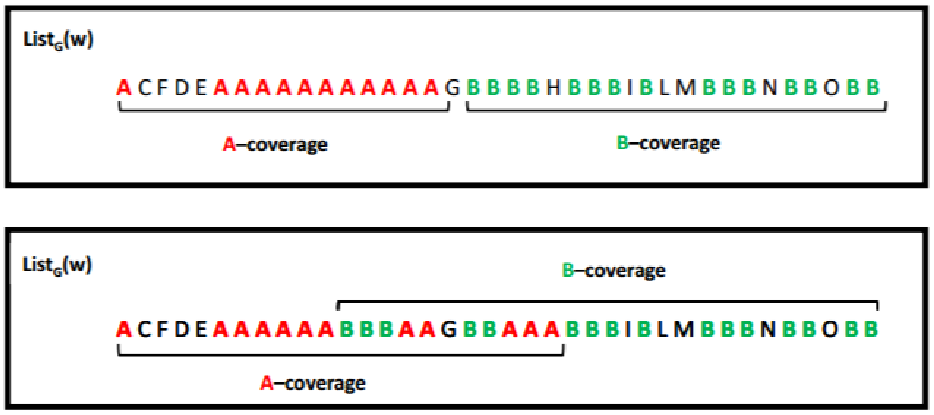

$List_G(w)$
A C F D E A A A A A A A A A A A G B B B B H B B B I B L M B B B N B B O B B
A–coverage
B–coverage
$List_G(w)$
B–coverage
A C F D E A A A A A A B B B A A G B B A A A B B B I B L M B B B N B B O B B
A–coverage

Figure 1: Example of what the g coverage of two fused genes (genes represented by letters) might look like.

4.2 Combinatory methods examined

To calculate the coverage measure, we considered three different combinatorial algorithms that calculate the fusion score for each dataset, based on the ranges of various gene pairs under consideration.

4.2.1 statistical analysis with known genes check-range-majority

1. The prediction file is loaded (predicted fingerprints for each read).

2. A preliminary check is performed to ensure that the two genes are actually present; otherwise, it moves on to the next gene pair.

3. **Two cases** of overlap between the ranges of the two genes are analyzed:

   (a) A check ensuring that the intervals of the two genes are well-defined, have non-negative length, and that the **first gene ends after the second**, avoiding unwanted overlaps or inconsistent situations in the intervals of the two genes.

   (b) A check ensuring that the intervals of the two genes are well-defined, have non-negative length, and that the **second gene ends before the first**, avoiding unwanted overlaps or inconsistent situations in the intervals of the two genes. The main difference from the previous check is the order of conditions, changing the orientation of the relationship between the two genes.

4. In either of the two previous cases, a check is carried out to ensure that the fusion score is greater than or equal to the adaptive threshold.

5. If all these conditions are met, metrics are calculated based on the input dataset (chimeric or non-chimeric), generating true positives, true negatives, false positives, and false negatives, summarized by the F1-score.

4.2.2 statistical analysis with known genes no-check-range-majority

1. The prediction file is loaded (predicted fingerprints for each read).

2. A preliminary check is performed to ensure that the two genes are actually present; otherwise, it moves on to the next gene pair.

3. It looks for the **most prevalent (or common) gene**, meaning the one that appears the most times within both the range of gene 1 and the range of gene 2.

4. A check is carried out to ensure that the fusion score is greater than or equal to the adaptive threshold.

5. If all these conditions are met, metrics are calculated based on the input dataset (chimeric or non-chimeric), generating true positives, true negatives, false positives, and false negatives, summarized by the F1-score.

4.2.3 statistical analysis with known-genes-consecutive-frequency

1. The prediction file is loaded (predicted fingerprints for each read).
2. A preliminary check is performed to ensure that the two genes are actually present; otherwise, it moves on to the next gene pair.
3. It looks for the **gene that reoccurs most consecutively**, meaning the one that appears the greatest number of times consecutively within both the range of gene 1 and the range of gene 2.
4. A check is carried out to ensure that the fusion score is greater than or equal to the adaptive threshold.
5. If all these conditions are met, metrics are calculated based on the input dataset (chimeric or non-chimeric), generating true positives, true negatives, false positives, and false negatives, summarized by the F1-score.

These combinatorial methods allow us to understand, for each gene panel, which one is the best based on the returned metrics, and consequently, implicitly, based on the number of chimeric and non-chimeric reads returned.

5 Tests Conducted

Tests were conducted on 2 gene panels, one containing all 100 genes, the other on a panel of 10 randomly selected genes, and the following procedures were performed:
First, fingerprints were generated from transcripts related to a gene dataset, then these were used to train the model with Random Forest.

5.1 Generation of merger datasets with Fusim

First, the Fusim tool was executed, which, given the **name of two or more genes and their connected transcripts**, the reference genome (in our case, **hg19.fa**), and the gene model **refFlat**, allows the generation of an arbitrary number of fused or **chimeric** transcripts (with nucleotide patterns from gene 1 and nucleotide patterns from gene 2) or **non-chimeric** transcripts if the two starting genes on which the algorithm is applied are the same. Subsequently, a script was used to convert all generated transcripts into a compatible format, creating a **.FASTQ** file.

Finally, the Random Forest model is inputted with the dataset of simulated Fusim fusions in the gene panel of arbitrary size, generating a prediction file named **test_fusion_result_CFL_ICFL_COMB-30_K8_dataset_name.txt**, where the factorizations on which we trained the model and predicted the fingerprints are based on $CFL - ICFL^d$**, with T = 20 or T = 30, which are the values that provided the best performance**.

5.2 Execution of combinatorial methods

We choose whether to execute one, two, or all three combinatorial methods based on testing needs. After executing the appropriate method, it will generate two fusion count files, indicating the number of occurrences for each gene in the dataset:

- **gene_fusion_count** : in the format gene_number-number of occurrences of that gene

  - 98:35
  - 98:28
  - 98:12
  - 92:82

- **parsed_gene_fusion_count** : in the format gene_name-number of occurrences of that gene

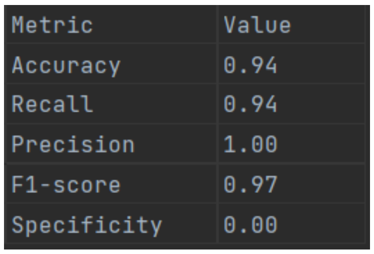
  - ENSG00000205129:35
  - ENSG00000205129:28
  - ENSG00000205129:12
  - ENSG00000197580:82

Subsequently, the method will load predictions from the file **test_fusion_result_CFL_ICFL_COMB-30_K8_dataset_name.txt** and perform calculations on gene ranges, ultimately computing a fusion score based on an adaptive threshold, as explained in Chapter 4 and specifically in 4.2, generating a file named **statistics_Method_dataset_name (chimeric or non-chimeric)** with information about the fusions for each read.

5.2.1 Results

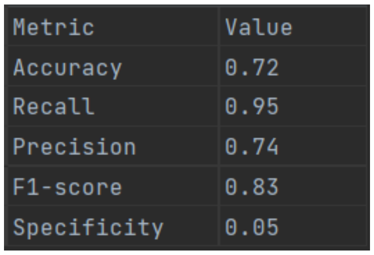
10 gene dataset

The test results for the 10-gene dataset yielded the following metrics for the 3 methods:



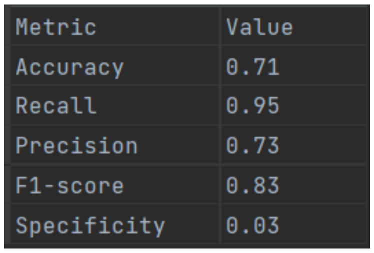
Metric	Value
Accuracy	0.94
Recall	0.94
Precision	1.00
F1-score	0.97
Specificity	0.00

(a) Check Range Majority.



Metric	Value
Accuracy	0.72
Recall	0.95
Precision	0.74
F1-score	0.83
Specificity	0.05

(b) Most Consecutive Frequency.



Metric	Value
Accuracy	0.71
Recall	0.95
Precision	0.73
F1-score	0.83
Specificity	0.03

(c) No Check Range Majority.

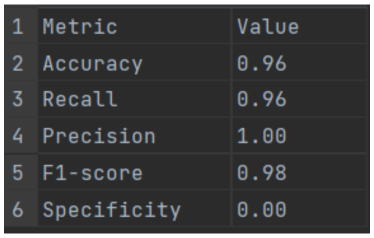
Figure 1: The three combinatorial methods applied to the 10-gene dataset.

Specifically, delving into the Check Range Majority method, we note that it provided an **optimal threshold of 0.1** and the following metrics: 165 True positives, 0 False positives, 0 True negatives, and 10 False negatives, identifying **165 chimeric transcripts** out of **218** transcripts that met the checks of the combinatorial methods and threshold (starting from 450 transcripts labeled chimeric and 100 labeled non-chimeric), minimizing False positives and False negatives.

This threshold is optimal because increasing it results in a less advantageous compromise in metrics, excessively reducing the number of chimeric reads. The other combinatorial methods (Most consecutive frequency and No check range Majority) in this case yield significantly lower results than Check Range Majority, emphasizing how the choice of method for each gene panel can make a difference in terms of results. Performing all three metrics on the gene panel provides an overall picture of the number of chimeric and non-chimeric reads.

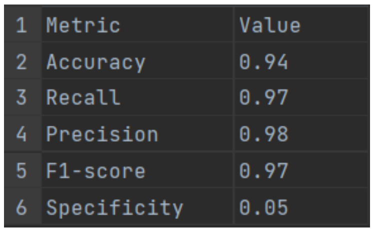
Dataset of 100 genes

The test results for the 100-gene dataset yielded the following metrics for the 3 methods:



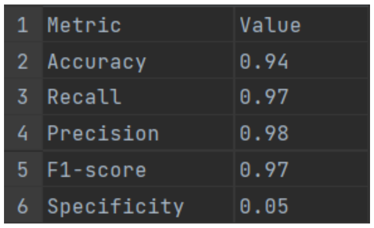
Metric	Value
Accuracy	0.96
Recall	0.96
Precision	1.00
F1-score	0.98
Specificity	0.00

(a) Check Range Majority.



Metric	Value
Accuracy	0.94
Recall	0.97
Precision	0.98
F1-score	0.97
Specificity	0.05

(b) Most Consecutive Frequency.



Metric	Value
Accuracy	0.94
Recall	0.97
Precision	0.98
F1-score	0.97
Specificity	0.05

(c) No Check Range Majority.

Figure 2: The three combinatorial methods applied to the 100-gene dataset.

Specifically, delving into the Check Range Majority method, we note that it provided an **optimal threshold of 0.1** and the following metrics: 28424 True positives, 0 False positives, 0 True negatives, and 1097 False negatives, identifying **28424 chimeric transcripts** out of **29521** transcripts that met the checks of the combinatorial methods and threshold (starting from 44652 transcripts labeled chimeric and 960 labeled non-chimeric), minimizing False positives and False negatives.

This threshold is optimal because, similar to the experiment with 10 genes, increasing it results in a less advantageous compromise in metrics, excessively reducing the number of chimeric reads. Compared to the experiment with 10 genes, the Check Range Majority method is only slightly better than the other combinatorial methods (Most consecutive frequency and No check range Majority), precisely by a single point in F1-score. Therefore, in this case, the choice of the Check Range Majority method does not have as much impact as in the previous experiment, so using the other two methods would not significantly worsen the results.

6 Conclusion and future developments

In conclusion, we can assert that we have succeeded in achieving the desired goal, namely, being able to threshold transcript datasets (long reads) in order to find that point (threshold) from which we can identify the maximum number of chimeric reads with an optimal compromise. Although the choice of an adaptive threshold is efficient and allows us to find excellent compromises, it can still be improved. We could choose it not only based on the F1-score value but also considering a certain number of chimeric and non-chimeric reads. Alternatively, we may prioritize the false positive metric over the false negative and vice versa, especially in cases where the two individual metrics are unbalanced enough to worsen the results.