

UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL CIENCIA DE LA COMPUTACIÓN



COMPUTACIÓN MOLECULAR BIOLÓGICA

Informe Toward and alignment-free method

ALUMNOS:

TORRES LIMA, JOSE
DIAZ VENTURA, NOEL
HUAYPUNA HUANCA, JOHANN
HUISA QUISPE, LUIS

DOCENTE:

Prof: MSc. VICENTE MACHACA ARCEDA

AREQUIPA - PERÚ

2020

ÍNDICE GENERAL

1	Introducción	2
2	Métodos con alineación y sin alineación	2
3	MÉTODO	2
3.1	Objetivos	2
3.2	Algoritmo CASTOR-KRFE	3
3.3	Conjunto de datos	4
4	Implementación	5
4.1	Pasos principales	5
4.1.1	Composición y preprocesamiento de vectores de características	5
4.1.2	Identificación de conjuntos de características y su evaluación	6
4.1.3	Extracción de los k-mers	6
4.2	Código Completo	6
5	Resultados	18
6	REPOSITORIO	19

1 INTRODUCCIÓN

La clasificación de secuencias genómicas es una práctica fundamental en diferentes campos de la investigación en microbiología. Consiste en asignar una secuencia dada a su grupo relacionado de secuencias conocidas que comparten características y rasgos similares.

CASTOR-KRFE es un método sin alineación que extrae características discriminantes de longitud óptima a partir de secuencias genómicas. Construye un modelo de predicción con una evaluación de éste. El modelo se puede utilizar para realizar la predicción de nuevas secuencias.

2 MÉTODOS CON ALINEACIÓN Y SIN ALINEACIÓN

Se implementaron varios métodos en herramientas para clasificar secuencias virales. Los métodos iniciales se basan en la alineación de secuencias que incluyen (1) herramientas de propósito general para la búsqueda de similitudes, como BLAST (Altschul et al., 1997) y USEARCH (Edgar, 2010) y (2) herramientas específicas de organismos como REGA (De Oliveira et al., 2005) y SCUEAL (Pond et al., 2009) para la clasificación del virus de inmunodeficiencia humana 1 (VIH-1). Sin embargo, el uso de métodos basados en alineación podría enfrentar ciertas limitaciones. De hecho, consumen mucho tiempo y memoria, y su carga computacional para grandes bases de datos presenta un cuello de botella real (Bonham-Carter et al., 2013; Zielezinski et al., 2017). Los métodos basados en alineación no son adecuados para genomas, como los de virus, que están expuestos a grandes variaciones genéticas como recombinación, mezcla y transferencia de genes horizontal (Duffy et al., 2008; Zielezinski et al., 2017). Además, realizar una alineación requiere a menudo el ajuste de varios parámetros (matrices de sustitución, penalizaciones por huecos, valores de umbral para parámetros estadísticos, etc.) que dependen del conocimiento a priori de la evolución de las secuencias que se comparan.

Por lo tanto, para superar estos problemas, los métodos sin alineación surgieron como una alternativa en la comparación y clasificación de secuencias. Transforman secuencias biológicas en vectores de características (Vinga, 2014) para calcular una distancia y construir un modelo filogenético o de aprendizaje automático (Xing et al., 2010). El manejo de secuencias en un espacio vectorial evita correspondencias de residuos, lo que aumenta la velocidad de cálculo y proporciona robustez para la predicción de la variación genética (Zielezinski et al., 2017). Para obtener una clasificación precisa utilizando enfoques sin alineación, es obligatorio identificar las características relevantes.

Hoy en día, los k-mers (subsecuencias de nucleótidos de longitud k) constituyen una alternativa o complementariedad para vectorizar secuencias de nucleótidos (Bonham-Carter et al., 2013). El uso de este tipo de atributo ha demostrado su eficacia en la clasificación de secuencias a través de árboles filogenéticos (Blaisdell, 1986; Sims et al., 2009; Kolekar et al., 2012), enfoques basados en la distancia (Otu y Sayood, 2003; Liu et al., 2011; Nalbantoglu et al., 2011), o métodos basados en estadísticas y composiciones de nucleótidos (Ulitsky et al., 2006; Reinert et al., 2009; Chan et al., 2012).

3 MÉTODO

3.1 OBJETIVOS

En este trabajo se presenta a CASTOR-KRFE (extracción de k-mers por eliminación de características recursivas) para construir un conjunto de características relevantes para la clasificación de secuencias genómicas. Con este método, esperamos lograr tres objetivos principales: primero, extraer un conjunto mínimo de discriminaciones de subsecuencias maximizando los rendimientos de clasificación viral; segundo, identificar la longitud óptima de las subsecuencias maximizando la precisión de la clasificación; finalmente, haga que estas características sean fácilmente interpretables a escala humana. De hecho, varias subsecuencias extraídas podrían corresponder a información biológica significativa.

3.2 ALGORITMO CASTOR-KRFE

El algoritmo de aprendizaje CASTOR-KRFE toma como entrada (1)S , un conjunto de n secuencias conocidas, tal que, donde s i es una cadena de caracteres de un alfabeto finito, y (2)y, un vector de etiquetas correspondientes a las clases de cada secuencia. k-mers se definen como las subsecuencias superpuestas de una secuencia s_i con longitudes de k. Ahora deja y ser, respectivamente, las longitudes mínima y máxima de k-mers y K el conjunto de longitudes a explorar de manera que el número de k-mers posibles está teóricamente limitado. El objetivo de CASTOR-KRFE es identificar el conjunto mínimo de k-mers con la longitud óptima de k maximizando la clasificación de cada secuencia de S dentro de las clases y según la puntuación de la medida F. El algoritmo CASTOR-KRFE se puede dividir en tres componentes principales. El primer componente es la composición y preprocesamiento de vectores de características. El segundo es la identificación de conjuntos de características y su evaluación. El tercero corresponde a la extracción de los k-mers que representan los conjuntos de características óptimos para construir un modelo de predicción.

En el algoritmo 1 , para cada uno de los conjuntos k-mers presentes en S y se calcula una matriz de ocurrencias X. X contiene, para cada secuencia, el número de apariciones de k-mers. A escala min max entre 0 y 1 se aplica entonces a X . Como se menciona en Hsu et al. (2003), permite evitar el dominio de características con rangos numéricos mayores sobre aquellas con rangos numéricos menores y disminuye las dificultades numéricas durante el cálculo. Además, la escala min-max ha demostrado su capacidad para lograr un mejor rendimiento que otros métodos de normalización (Al Shalabi et al., 2006).

Después de este paso de preprocesamiento, el algoritmo comenzará analizando si el número actual de características (corresponde al número de columnas en la matriz X) es mayor que el número máximo de características dado en el parámetro (). Si este es el caso, se llama a una selección preliminar con eliminación de características recursivas basada en la máquina de vectores de soporte (SVM-RFE) (Guyon et al., 2002). La clasificación se basa en los pesos asignados al clasificador. Para cada iteración, se elimina un número determinado de funciones hasta alcanzar el número de funciones objetivo. Para este SVM-RFE preliminar, se elige un paso de eliminación predeterminado del 10% para eliminar rápidamente la mayoría de las características innecesarias y retener solo características. Una vez que se completa este paso preliminar, para cada f que van desde a, SVM-RFE se aplicará para reducir el número de características a f mediante un paso de eliminación de 1. Cada conjunto de características seleccionado se evaluará mediante una validación cruzada de pliegues x estratificada, métricas de rendimiento (medida F ponderada) y una clasificador (SVM), donde x se puede dar como entrada.

Algorithm 1: CASTOR-KRFE: Features extractor

```

Input: S: labeled nucleotide sequences,
       $k_{min}$ : minimum length of k to identify,
       $k_{max}$ : maximum length of k to identify,
       $f_{min}$ : minimum number of features,
       $f_{max}$ : maximum number of features,
Output:  $f_{list}$  list of potential feature set,
        $s_{list}$ : scores assigned to  $f_{list}$ 

1 Begin
2    $f_{list} \leftarrow \emptyset$ 
3    $s_{list} \leftarrow \emptyset$ 
4   foreach  $k \in [k_{min} \dots k_{max}]$  do
5      $D \leftarrow k\text{-mers} \in S$ 
6     foreach  $s_i \in S$  do
7        $X \leftarrow \text{occurrences of each } k\text{-mers} \in D$ 
8     end
9      $X \leftarrow \text{MinMaxScaler}(X, 0, 1)$ 
10    if  $\text{number\_of\_features}(X) > f_{max}$  then
11       $X \leftarrow \text{RFE}(X, f_{max})$            #Apply RFE to obtain  $f_{max}$  features
12    foreach  $f \in [f_{min} \dots f_{max}]$  do
13       $X \leftarrow \text{RFE}(X, f)$ 
14       $f_{list} \leftarrow \text{features}(X)$ 
15       $s_{list} \leftarrow \text{Cross-Validation}(X)$ 
16    end
17  end
18 End

```

Figure 3.1: LEBATTEUX ET AL 2019

Algorithm 2: CASTOR-KRFE: Optimal set identifier and prediction model builder

Input: f_{list} : list of extracted feature sets,
 s_{list} : scores assigned to f_{list} ,
 T : percentage performance threshold
Algorithm: supervised learning algorithm
Output: k_{length} : optimal length of k ,
 f_{set} : optimal set of k -mers,
model: predictive model based on f_{set}

```
1 Begin
2    $best_{\text{score}} \leftarrow \max(s_{\text{list}})$ 
3    $f_{\text{set}} \leftarrow \text{feature set } \in f_{\text{list}} \text{ associated with } best_{\text{score}}$ 
4    $n \leftarrow \text{len}(f_{\text{set}})$ 
5    $k_{\text{length}} \leftarrow \text{len}(\text{features of } f_{\text{set}})$ 
6   foreach  $f, s \in f_{\text{list}}, s_{\text{list}}$  do
7     if  $s \geq best_{\text{score}} * T$  and  $\text{len}(f) < n$  then
8        $f_{\text{set}} \leftarrow f$ 
9        $n \leftarrow \text{len}(f)$ 
10       $k_{\text{length}} \leftarrow \text{len}(\text{features of } f)$ 
11    end
12  end
13   $model \leftarrow (\text{Algorithm}, f_{\text{set}})$ 
14 End
```

Figure 3.2: LEBATTEUX ET AL 2019

Los conjuntos de características y sus puntuaciones asociadas se incluirán en los extractores de características y en la evaluación del clasificador como se describe en el algoritmo 2. El algoritmo utiliza una SVM basada en un kernel lineal. Este núcleo requiere unos pocos hiperparámetros que implican una baja complejidad del modelo de selección. En la última parte del algoritmo 2, analizamos las listas de puntuación de rendimiento para extraer el conjunto de características óptimas y la longitud k óptima de las subsecuencias. La característica óptima seleccionada debe satisfacer dos condiciones. Primero, tiene una puntuación de medida F más alta que la mejor puntuación multiplicada por el umbral de rendimiento T (porcentaje de rendimiento mantenido en relación con la mejor puntuación). En segundo lugar, tiene la menor cantidad de funciones. Finalmente, se entrena un algoritmo de aprendizaje supervisado con el conjunto de características óptimas para construir un modelo predictivo.

3.3 CONJUNTO DE DATOS

Se aplica el algoritmo a un conjunto heterogéneo de datos genómicos que cubren los siete principales grupos de virus. De una amplia gama de conjuntos de datos, tomamos muestras de virus reemergentes que afectan a las poblaciones mundiales actuales, como el virus del Ébola, el VIH, el virus de la hepatitis C (VHC) y el virus del dengue. La identificación precisa y rápida de este tipo de virus puede tener importantes impactos para su estudio y la preservación de la vida y la salud humanas. De hecho, la enfermedad por el virus del Ébola se asocia con una tasa de letalidad del 30% al 90%, según la especie del virus (Baize et al., 2014). El virus del dengue también se ha incluido en nuestro conjunto de datos. La fiebre del dengue es una enfermedad emergente en todas las regiones tropicales y subtropicales de Asia, África, América y el Pacífico (Vaughn et al., 2000). Cada año, se estiman al menos 100 millones de infecciones por dengue (Halstead, 1988). La infección por dengue se asocia con fiebre hemorrágica y síndromes de choque, que es una de las principales causas de morbilidad y mortalidad pediátricas en Asia tropical (sangkawibha et al., 1984).

Algunos virus reemergentes, al mismo tiempo, son complejos y con una gran variedad de subtipos y altas tasas de recombinación. De hecho, el VHC incluye muchos genotipos que difieren entre sí en aproximadamente un 30% a nivel de nucleótidos. Además, cada genotipo está compuesto por múltiples subtipos divergentes de 20% en su composición de nucleótidos (Simmonds et al., 2005). La clasificación del VIH también representa un gran interés. De hecho, su grupo predominante M se divide actualmente en 13 subtipos. La variación genética dentro de un subtipo puede oscilar entre el 15% y el 20%, mientras que la variación entre subtipos suele ser del 25% al 35% (Taylor et al., 2008). Además, el VIH tiene altas tasas de mutación y recombinación. Son dos características esenciales del ciclo de replicación que permiten la propagación continua del VIH en todo el mundo, lo que implica una pandemia de complejidad genética y geográfica sin precedentes (Taylor et

al., 2008).

4 IMPLEMENTACIÓN

Se establecen los parametros para los calculos.

```
1
2
3 # VARIABLES
4
5 # Umbral (porcentaje de prdida de rendimiento en trminos de medida F para reducir el
  ↪ nmero de atributos)
6 T = 0.999
7 # Longitud mnima de k-mer(s)
8 k_min = 1
9 # Longitud mxima de k-mer(s)
10 k_max = 5
11 # Nmero mnimo de caractersticas para identificar
12 features_min = 1
13 # Nmero maximo de caractersticas para identificar
14 features_max = 100
15 # Ruta de archivo fasta de entrenamiento
16 training_fasta = "Input/HPV_Alpha_species_sample_1.fa"#"Input/HIVGRPCG/data.fasta"
17 # Ruta de archivo fasta de entrenamiento
18 training_csv = "Input/HPV_Alpha_species_sample_1.csv"#"Input/HIVGRPCG/target.csv"
19 # Ruta de archivo fasta de pruebas
20 testing_fasta = "Input/HPV_Alpha_species_sample_1.fa"#"Input/HIVGRPCG/data.fasta"
21 # Ruta de archivo fasta de pruebas
22 testing_csv = "Input/HPV_Alpha_species_sample_1.csv"#"Input/HIVGRPCG/target.csv"
```

4.1 PASOS PRINCIPALES

4.1.1 COMPOSICIÓN Y PREPROCESAMIENTO DE VECTORES DE CARACTERÍSTICAS

En esta sección se realiza la comprobación de la existencia y accesibilidad de archivos de entrenamiento(archivo fasta) y de las etiquetas(archivos csv) como también de los archivos de pruebas. Se da el inicio de la extracción de características para k-mers. Ademas de la vectorización de secuencias genómicas virales conocidas basadas en k-mers para constituir las características potenciales. Y la generación del dato de entrenamiento.

Listing 1: Comprobación de Datos

```
1
2
3 # CARGAR DATOS DE ENTRENAMIENTO
4 print("\nLoading of the training dataset...")
5 if Data.checkTrainFile(training_fasta, training_csv) == True: training_data = Data.generateTrainData(
  ↪ training_fasta, training_csv)
6
7
8 # EXTRACCIN DE CARACTERSTICAS
9 print("\nStart feature extraction...")
10 extracted_k_mers, identified_k_length = Extraction.extractKmers(T, training_data, k_min, k_max,
  ↪ features_min, features_max)
```

4.1.2 IDENTIFICACIÓN DE CONJUNTOS DE CARACTERÍSTICAS Y SU EVALUACIÓN

En esta sección se realiza la evaluación del modelo, mediante prueba cruzada y se cargan los datos de prueba para medir la precisión del clasificador. Forma eficiente de extracción y evaluación de patrones que maximizan el rendimiento de clasificación

```
1
2
3 # EVALUACION DEL MODELO
4 print("\nEvaluation of the prediction model...")
5 Evaluation.cross_validation(training_data, extracted_k_mers, identified_k_length, training_data)
6
7
8 # CARGAR DATOS DE PRUEBA
9 print("\nLoading of the testing dataset...")
10 if Data.checkTestFile(testing_fasta, testing_csv) == True: testing_data = Data.generateTestData(
    ↪ testing_fasta, testing_csv)
```

4.1.3 EXTRACCIÓN DE LOS K-MERS

Predicción del conjunto mínimo de características que se ajustan a un criterio dado (umbral de métrica de rendimiento y número máximo de características). Se obtiene el valor de longitud optimo para K.

```
1
2
3 # PREDICCION
4 if len(testing_data[0]) == 2:
5     print("\nPrediction without evaluation...")
6     Evaluation.prediction(training_data, testing_data, extracted_k_mers, identified_k_length)
7 else:
8     print("\nPrediction with evaluation...")
9     Evaluation.predictionEvaluation(training_data, testing_data, extracted_k_mers, identified_k_length
    ↪ )
10
11
12 print("\nEnd of the program ")
```

4.2 CODIGO COMPLETO

```
1
2 # IMPORTS
3 import Data
4 import Extraction
5 import Evaluation
6
7 # INFORMACION
8 print("*****")
9 print("*** CASTOR-KRFE ***")
10 print("*****\n")
11
12 print("Alignment-free method to extract discriminant genomic subsequences within pathogen
    ↪ sequences.\n")
13
14 # VARIABLES
```

```

15 |
16 | # Umbral (porcentaje de prdida de rendimiento en trminos de medida F para reducir el
    |     ↪ nmero de atributos)
17 | T = 0.999
18 | # Longitud mnima de k-mer(s)
19 | k_min = 1
20 | # Longitud mxima de k-mer(s)
21 | k_max = 5
22 | # Nmero mnimo de caractersticas para identificar
23 | features_min = 1
24 | # Nmero maximo de caractersticas para identificar
25 | features_max = 100
26 | # Ruta de archivo fasta de entrenamiento
27 | training_fasta = "Input/HPV_Alpha_species_sample_1.fa"#"Input/HIVGRPCG/data.fasta"
28 | # Ruta de archivo fasta de entrenamiento
29 | training_csv = "Input/HPV_Alpha_species_sample_1.csv"#"Input/HIVGRPCG/target.csv"
30 | # Ruta de archivo fasta de pruebas
31 | testing_fasta = "Input/HPV_Alpha_species_sample_1.fa"#"Input/HIVGRPCG/data.fasta"
32 | # Ruta de archivo fasta de pruebas
33 | testing_csv = "Input/HPV_Alpha_species_sample_1.csv"#"Input/HIVGRPCG/target.csv"
34 |
35 |
36 | # CARGAR DATOS DE ENTRENAMIENTO
37 | print("\nLoading of the training dataset...")
38 | if Data.checkTrainFile(training_fasta, training_csv) == True: training_data = Data.generateTrainData(
    |     ↪ training_fasta, training_csv)
39 |
40 |
41 | # EXTRACCIN DE CARACTERSTICAS
42 | print("\nStart feature extraction...")
43 | extracted_k_mers, identified_k_length = Extraction.extractKmers(T, training_data, k_min, k_max,
    |     ↪ features_min, features_max)
44 |
45 |
46 | # EVALUACIN DEL MODELO
47 | print("\nEvaluation of the prediction model...")
48 | Evaluation.cross_validation(training_data, extracted_k_mers, identified_k_length, training_data)
49 |
50 |
51 | # CARGAR DATOS DE PRUEBA
52 | print("\nLoading of the testing dataset...")
53 | if Data.checkTestFile(testing_fasta, testing_csv) == True: testing_data = Data.generateTestData(
    |     ↪ testing_fasta, testing_csv)
54 |
55 |
56 | # PREDICCIÓN
57 | if len(testing_data[0]) == 2:
58 |     print("\nPrediction without evaluation...")
59 |     Evaluation.prediction(training_data, testing_data, extracted_k_mers, identified_k_length)
60 | else:
61 |     print("\nPrediction with evaluation...")
62 |     Evaluation.predictionEvaluation(training_data, testing_data, extracted_k_mers, identified_k_length
    |     ↪ )
63 |

```



```

64
65 print("\nEnd of the program ")

1 # Imports
2 import os
3 import sys
4 import csv
5 from Bio import SeqIO
6
7 # Funcin de comprobacin de la existencia y accesibilidad de archivos de formacin.
8 def checkTrainFile(training_fasta, training_csv):
9     # Ver archivo fasta
10    if os.path.isfile(training_fasta) and os.access(training_fasta, os.R_OK):
11        print(training_fasta, "file exists and is readable")
12    # Comprobar archivo csv
13    if os.path.isfile(training_csv) and os.access(training_csv, os.R_OK):
14        print(training_csv, "file exists and is readable")
15    # Devuelve verdadero si todo es correcto
16    return True
17    # Salir y mostrar un mensaje en caso de error
18    else: sys.exit("Training csv file is missing or not readable")
19 else: sys.exit("Training fasta file is missing or not readable")
20
21 # Funcin que comprueba la existencia y accesibilidad de los archivos de prueba.
22 def checkTestFile(testing_fasta, testing_csv):
23     # Ver archivo fasta
24    if os.path.isfile(testing_fasta) and os.access(testing_fasta, os.R_OK):
25        print(testing_fasta, "file exists and is readable")
26    # Comprobar archivo csv
27    if os.path.isfile(testing_csv) and os.access(testing_csv, os.R_OK):
28        # Devuelve True si todo es correcto (prediccin con evaluacin)
29        print(testing_csv, "file exists and is readable")
30        return True
31    else:
32        # Devuelve True si solo el archivo fasta es correcto (prediccin sin evaluacin)
33        print("Testing csv file is missing or not readable")
34        return True
35    else: sys.exit("Testing fasta file is missing or not readable")
36
37 # Funcin que genera la tabla de datos
38 def generateTrainData(fasta_file, csv_file):
39     # Variable data
40     data = []
41
42     # Abrir el archivo de la clase
43     with open(csv_file) as f: reader = dict(csv.reader(f))
44
45     #Abrir el archivo de secuencias
46     for record in SeqIO.parse(fasta_file, "fasta"):
47         # Generar tabla [Id, Sequences, Class]
48         if record.id in reader: data.append([record.id, record.seq.upper(), reader[record.id]])
49
50     # Return data
51     return data

```

```

52
53 # Funcin que genera la tabla de datos
54 def generateTestData(fasta_file, csv_file):
55     # Variable data
56     data = []
57
58     # Lllamar a la funcin clsica
59     if csv_file: data = generateTrainData(fasta_file, csv_file)
60     else:
61         # Abra el archivo de secuencias y genere la tabla [Id, Sequences]
62         for record in SeqIO.parse(fasta_file, "fasta"): data.append([record.id, record.seq.upper()])
63
64     # Return data
65     return data

```

```

1 # Imports
2 import joblib
3 import Matrices
4 import Preprocessing
5 from sklearn import svm
6 from sklearn.metrics import confusion_matrix
7 from sklearn.metrics import classification_report
8 from sklearn.model_selection import StratifiedKFold
9 from sklearn.model_selection import cross_val_predict
10
11 # Funcin de evaluacin del modelo
12 def cross_validation(training_data, extracted_k_mers, identified_k_length, data):
13     # Generar matrices
14     X, y = Matrices.generateMatrice(training_data, extracted_k_mers, identified_k_length)
15     X = Preprocessing.minMaxScaling(X)
16
17     # Realice la evaluacin con CV + Clasificador + Mtricas
18     classifier = svm.SVC(kernel = 'linear', C = 1)
19     stratifiedKFold = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)
20     y_pred = cross_val_predict(classifier, X, y, cv = stratifiedKFold, n_jobs = 4)
21
22     # Imprimir resultados de la evaluacin del modelo
23     classificationReport = classification_report(y, y_pred, digits = 3)
24     confusionMatrix = confusion_matrix(y, y_pred)
25     print("\nClassification report of model evaluation\n", classificationReport)
26     print("Confusion matrix \n", confusionMatrix)
27
28     # Guardar Matrice
29     f = open("Output/Matrice.csv", "w")
30     f.write("Id,")
31     for i in extracted_k_mers: f.write(str(i) + ",");
32     f.write("Class\n")
33
34     for i, x in enumerate(X):
35         f.write(str(data[i][0]) + ",")
36         for j in x: f.write(str(j) + ",")
37         f.write(str(y[i]) + "\n")
38     f.close()
39

```

```

40 # Guardar model
41 classifier.fit(X, y)
42 joblib.dump(classifier, 'Output/model.pkl')
43
44 # Guardar los resultados de la evaluacin del modelo
45 f = open("Output/Model_Evaluation.txt", "w")
46 f.write("Classification report of model evaluation\n" + classificationReport);
47 f.write("\nConfusion matrix \n" + str(confusionMatrix));
48 f.close()
49
50
51
52 # Funcin de prediccin sin evaluacin
53 def prediction(training_data, testing_data, extracted_k_mers, identified_k_length):
54     # Generar matrices
55     X_test, y_test = Matrices.generateMatrice(testing_data, extracted_k_mers, identified_k_length)
56     X_test = Preprocessing.minMaxScaling(X_test)
57
58     # Cargar model
59     classifier = joblib.load('Output/model.pkl')
60
61     # Realizar prediccin
62     y_pred = classifier.predict(X_test)
63
64     # Guardar prediccin
65     f = open("Output/Prediction.csv", "w")
66     f.write("id,y_pred\n");
67     for i, y in enumerate(y_pred): f.write(testing_data[i][0] + "," + y + "\n");
68     f.close()
69
70 # Funcin de prediccin con evaluacin
71 def predictionEvaluation(training_data, testing_data, extracted_k_mers, identified_k_length):
72     # Generar matrices
73     X_test, y_test = Matrices.generateMatrice(testing_data, extracted_k_mers, identified_k_length)
74     X_test = Preprocessing.minMaxScaling(X_test)
75
76     # Cargar model
77     classifier = joblib.load('Output/model.pkl')
78
79     # Realizar prediction
80     y_pred = classifier.predict(X_test)
81
82     # Imprimir resultados
83     classificationReport = classification_report(y_test, y_pred, digits = 3)
84     confusionMatrix = confusion_matrix(y_test, y_pred)
85     print("\nClassification report of prediction evaluation\n", classificationReport)
86     print("Confusion matrix \n", confusionMatrix)
87
88     # Guardar prediccin
89     f = open("Output/Prediction_Evaluation.csv", "w")
90     f.write("id,y_pred,y_true\n");
91     for i, y in enumerate(y_pred): f.write(testing_data[i][0] + "," + y + "," + y_test[i] + "\n");
92     f.close()
93

```

```

94 # Guardar los resultados de la evaluacin de la prediccin
95 f = open("Output/Prediction_Evaluation.txt", "w")
96 f.write("Classification report of prediction evaluation\n" + classificationReport);
97 f.write("\nConfusion matrix \n" + str(confusionMatrix));
98 f.close()

```

```

1 # Imports
2 import numpy
3 import K_mers
4 import Matrices
5 import Preprocessing
6 from sklearn import svm
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import f1_score
9 from sklearn.feature_selection import RFE
10 from sklearn.model_selection import StratifiedKFold
11 from sklearn.model_selection import cross_val_predict
12
13 # Funcin bsica de extraccin de caractersticas
14 def extractKmers(T, training_data, k_min, k_max, features_min, features_max):
15     # Contiene listas de puntuaciones de diferentes longitudes de k
16     scores_list = []
17     # Contiene una lista de k-mer para cada iteracin de rfe
18     supports_list = []
19     # Lista de diferentes longitudes de k-mer
20     k_mers_range = range(k_min, k_max + 1)
21     # Clasificador svm
22     classifier = svm.SVC(kernel = 'linear', C = 1)
23
24     # Realizar el anlisis para los diferentes tamaos de k
25     for k in k_mers_range:
26         # Iniciar extraccin de caractersticas basada en k-mers de longitud k
27         print("\nBeginning of the " + str(k) + "_mer(s) analysis")
28
29         # Generarte lista de k-mer
30         print("Generate K-mers...")
31         k_mers = K_mers.generate_K_mers(training_data, k)
32
33         # Generar atributos de matrices y clases de matrices
34         print("Generate matrices...")
35         X, y = Matrices.generateMatrice(training_data, k_mers, k)
36         y = numpy.asarray(y)
37
38         # Aplicar MinMaxScaler (0, 1)
39         X = Preprocessing.minMaxScaling(X)
40
41         # Si hay ms de features_max, aplique RFE (elimine el 10% de las caractersticas para
42         ↪ eliminar en cada iteracin)
43         if len(X[0]) > features_max:
44             print("Preliminary recursive feature elimination...")
45             rfe = RFE(estimator = classifier, n_features_to_select = features_max, step = 0.1)
46             X = numpy.matrix(X)
47             X = rfe.fit_transform(X, y)

```

```

48         # Actualizar lista de k_mers
49         for i, value in enumerate(rfe.support_):
50             if value == False: k_mers[i] = None
51         k_mers = list(filter(lambda a: a != None, k_mers))
52
53     # Eliminacin de caractersticas recursivas
54     from RFE import RFE
55     print("Recursive feature elimination...")
56     rfe = RFE(estimator = classifier, n_features_to_select = 1, step = 1)
57     rfe.fit(X,y)
58
59     # Puntajes y apoyos de la iteracin actual
60     scores = []
61     supports = []
62
63     # Evaluacin
64     for i, supports_rfe in enumerate(rfe.supports):
65         # Variables
66         temp_index = []
67         temp_k_mers = []
68
69         # Imprimir porcentaje de avance
70         print("\rFeature subset evaluation :", round((i + 1) / len(rfe.supports) * 100, 0), "%",
71               ↪ end = '')
72
73         # Selecciona k-mers con soporte igual True
74         for j, support in enumerate(supports_rfe):
75             if rfe.supports[i][j] == True: temp_index.append(j)
76
77         # Reemplazar el soporte por los k-mers
78         for t in temp_index: temp_k_mers.append(k_mers[t])
79         rfe.supports[i] = temp_k_mers
80
81         # Mtodo de evaluacin
82         stratifiedKFold = StratifiedKFold(n_splits = 5, shuffle = False, random_state = None)
83         y_pred = cross_val_predict(classifier, X[:,temp_index], y, cv = stratifiedKFold, n_jobs = 4)
84         score = f1_score(y, y_pred, average = 'weighted')
85
86         # Guardar la puntuacin y las caractersticas de esta iteracin
87         scores.append(score)
88         supports.append(rfe.supports[i])
89
90     # Guarde la lista de puntuaciones y subconjuntos de funciones para esta longitud de
91     ↪ k-mers
92     scores_list.append(scores)
93     supports_list.append(supports)
94
95     # Cambia el orden de las listas para el grfico.
96     for i, e in enumerate(scores_list):
97         scores_list[i].reverse()
98         supports_list[i].reverse()
99
100     # Identificar solucin
101     print("\n\nIdentify optimal solution...")

```

```

100 # Mejor puntuacin de las evaluaciones
101 best_score = 0
102 # Puntuacin ptima en relacin con el treshold
103 optimal_score = 0
104 # Mejor lista de k-mer
105 extracted_k_mers = []
106 # Mejor longitud de k
107 identified_k_length = 0
108
109 # Identificar la mejor solucin
110 for i, s in enumerate(scores_list):
111     if max(s) > best_score:
112         best_score = max(s)
113         index = s.index(max(s))
114         identified_k_length = k_mers_range[i]
115         extracted_k_mers = supports_list[i][index]
116     elif max(s) == best_score:
117         if s.index(max(s)) < index:
118             best_score = max(s)
119             index = s.index(max(s))
120             identified_k_length = k_mers_range[i]
121             extracted_k_mers = supports_list[i][index]
122     else: pass
123
124 # Identificar la solucin ptima
125 for i, l in enumerate(scores_list):
126     for j, s in enumerate(l):
127         if s >= best_score * T and j <= index:
128             optimal_score = s
129             index = j
130             identified_k_length = k_mers_range[i]
131             extracted_k_mers = supports_list[i][index]
132 if optimal_score == 0: optimal_score = best_score
133
134
135 # Guardar los resultados del grfico
136 fig = plt.figure(figsize = (12, 10) )
137 for i, s in enumerate(scores_list):
138     label = str(k_mers_range[i]) + "-mers"
139     plt.plot(range(len(s)), s, label = label)
140 plt.ylabel("F-measure")
141 plt.xlabel("Number of features")
142 plt.axvline(index + 1, linestyle = ':', color = 'r')
143 title = "F-measure : " + str(round(optimal_score, 3)) + " K-mer size : " + str(
    ↪ identified_k_length) + " Number of features : " + str(index + 1)
144 plt.title(title)
145 plt.legend()
146 fname = str("Output/Analysis.png")
147 plt.savefig(fname)
148
149 # Guardar k-mers extrados
150 f = open("Output/Kmers.txt", "w")
151 for i in extracted_k_mers: f.write(str(i) + "\n");
152 f.close()

```

```

153
154     # Devuelve k-mers identificados y su longitud
155     return extracted_k_mers, identified_k_length

1  import re #se importa la libreria para trabajar con expresiones regulares
2
3  # Funcin que genera los k_mers pertenecientes a las secuencias
4  def generate_K_mers(data, k):
5      # List of k-mer
6      K_mers = []
7      dict = {}
8
9      # Inicializacin del diccionario
10     for d in data:
11         for i in range(0, len(d[1]) - k + 1, 1): dict[d[1][i:i + k]] = 0;
12
13     # Eliminar patrones no utilizados
14     for key in dict.keys():
15         if bool(re.match('^[ACGT]+$ ', str(key))) == True: K_mers.append(str(key))
16
17     # Retorna los kmers
18     return K_mers

1  # Funcin de generacin de matrices de clases y caractersticas
2  def generateMatrice(data, K_mer, k):
3      # Variables
4      X = []
5      y = []
6
7      # Generar diccionario K-mer
8      X_dict = {}
9      for i, e in enumerate(K_mer): X_dict[e] = 0;
10     # Generar X (atributos de matriz)
11     for d in data:
12         x = []
13         x_dict = X_dict.copy()
14
15         # Contar ocurrencias de K-mer (con superposicin)
16         for i in range(0, len(d[1]) - k + 1, 1):
17             try: x_dict[d[1][i:i + k]] = x_dict[d[1][i:i + k]] + 1;
18             except: pass
19
20         # Obtener todas las ocurrencias del diccionario
21         for value in x_dict:
22             x.append(x_dict.get(value))
23         X.append(x)
24
25     # Genera y (clase Matrix) si existe un archivo csv
26     if len(data[0]) == 3:
27         for i in data: y.append(i[2])
28
29     # Retornar matrices X e y (atributos de matriz y clase de matriz)
30     return X, y

```

```

1 from sklearn.preprocessing import MinMaxScaler
2
3 # Funcion ode MinMaxScaler (0, 1)
4 def minMaxScaling(X):
5     minMaxScaler = MinMaxScaler(feature_range = (0, 1), copy = False)
6     X = minMaxScaler.fit_transform(X)
7     return X

```

```

1 #####
2 # Authors: Alexandre Gramfort <alexandre.gramfort@inria.fr> #
3 # Vincent Michel <vincent.michel@inria.fr> #
4 # Gilles Louppe <g.louppe@gmail.com> #
5 # #
6 # Modified by Dylan Lebatteux <lebatteux.dylan@courrier.uqam.ca> #
7 # #
8 # License: BSD 3 clause #
9 #####
10
11 # Imports
12 import numpy as np
13 from sklearn.base import clone
14 from sklearn.utils import safe_sqr
15 from sklearn.metrics import fl_score
16 from sklearn.base import BaseEstimator
17 from sklearn.base import is_classifier
18 from sklearn.metrics import check_scoring
19 from sklearn.base import MetaEstimatorMixin
20 from sklearn.model_selection import check_cv
21 from sklearn.utils.metaestimators import _safe_split
22 from sklearn.utils.validation import check_is_fitted
23 from sklearn.model_selection._validation import _score
24 from joblib import Parallel, delayed, effective_n_jobs
25 from sklearn.feature_selection.base import SelectorMixin
26 from sklearn.utils.metaestimators import if_delegate_has_method
27 from sklearn.utils.validation import _deprecate_positional_args
28
29 def _rfe_single_fit(rfe, estimator, X, y, train, test, scorer):
30     X_train, y_train = _safe_split(estimator, X, y, train)
31     X_test, y_test = _safe_split(estimator, X, y, test, train)
32     return rfe._fit(X_train, y_train, lambda estimator, features: _score(estimator, X_test[:, features],
33     ↪ y_test, scorer)).scores_
34
35 class RFE(SelectorMixin, MetaEstimatorMixin, BaseEstimator):
36     @_deprecate_positional_args
37     def __init__(self, estimator, *, n_features_to_select=None, step=1, verbose=0):
38         self.supports = []
39         self.estimator = estimator
40         self.n_features_to_select = n_features_to_select
41         self.step = step
42         self.verbose = verbose
43
44     @property
45     def _estimator_type(self): return self.estimator._estimator_type

```



```

45
46 @property
47 def classes_(self): return self.estimator_.classes_
48
49 def fit(self, X, y): return self._fit(X, y)
50
51 def _fit(self, X, y, step_score=None):
52     tags = self._get_tags()
53     X, y = self._validate_data(X, y, accept_sparse="csc", ensure_min_features=2, force_all_finite
        ↪ =not tags.get('allow_nan', True), multi_output=True)
54
55     # Inicializacin
56     n_features = X.shape[1]
57     if self.n_features_to_select is None: n_features_to_select = n_features // 2
58     else: n_features_to_select = self.n_features_to_select
59
60     if 0.0 < self.step < 1.0: step = int(max(1, self.step * n_features))
61     else: step = int(self.step)
62     if step <= 0: raise ValueError("Step must be >0")
63
64     support_ = np.ones(n_features, dtype=np.bool)
65     ranking_ = np.ones(n_features, dtype=np.int)
66
67     if step_score: self.scores_ = []
68
69     # Eliminacin
70     while np.sum(support_) > n_features_to_select:
71         # Features restantes
72         features = np.arange(n_features)[support_]
73
74         # Clasifique las caractersticas restantes
75         estimator = clone(self.estimator)
76         if self.verbose > 0: print("Fitting estimator with %d features." % np.sum(support_))
77
78         # Ajuste
79         estimator.fit(X[:, features], y)
80
81     # Obtener coefs
82     if hasattr(estimator, 'coef_'): coefs = estimator.coef_
83     else: coefs = getattr(estimator, 'feature_importances_', None)
84     if coefs is None: raise RuntimeError("The classifier does not expose coef_or
        ↪ feature_importances_attributes")
85
86     # Obtener rangos
87     if coefs.ndim > 1: ranks = np.argsort(safe_sqr(coefs).sum(axis=0))
88     else: ranks = np.argsort(safe_sqr(coefs))
89
90     # Para rangos de casos dispersos es matriz
91     ranks = np.ravel(ranks)
92
93     # Elimina las peores caractersticas
94     threshold = min(step, np.sum(support_) - n_features_to_select)
95
96     # Guardar soporte de caractersticas seleccionadas

```

```

97         self.supports.append(list(support_))
98
99         # Calcule el puntaje de paso en la iteracin de seleccin anterior porque el '
100         ↪ estimador' debe usar caractersticas que an no se han eliminado
101         if step_score: self.scores_.append(step_score(estimator, features))
102         support_[features[ranks][:threshold]] = False
103         ranking_[np.logical_not(support_)] += 1
104
105         # Establecer atributos finales
106         features = np.arange(n_features)[support_]
107         self.estimator_ = clone(self.estimator)
108         self.estimator_.fit(X[:, features], y)
109
110         # Calcule la puntuacin por pasos cuando solo quedan n caractersticas para
111         ↪ seleccionar caractersticas
112         if step_score: self.scores_.append(step_score(self.estimator_, features))
113         self.n_features_ = support_.sum()
114         self.support_ = support_
115         self.ranking_ = ranking_
116
117         # Guardar soporte de caractersticas seleccionadas
118         self.supports.append(list(support_))
119
120         return self
121
122     @if_delegate_has_method(delegate='estimator')
123     def predict(self, X):
124         check_is_fitted(self)
125         return self.estimator_.predict(self.transform(X))
126
127     @if_delegate_has_method(delegate='estimator')
128     def score(self, X, y):
129         check_is_fitted(self)
130         return self.estimator_.score(self.transform(X), y)
131
132     def _get_support_mask(self):
133         check_is_fitted(self)
134         return self.support_
135
136     @if_delegate_has_method(delegate='estimator')
137     def decision_function(self, X):
138         check_is_fitted(self)
139         return self.estimator_.decision_function(self.transform(X))
140
141     @if_delegate_has_method(delegate='estimator')
142     def predict_proba(self, X):
143         check_is_fitted(self)
144         return self.estimator_.predict_proba(self.transform(X))
145
146     @if_delegate_has_method(delegate='estimator')
147     def predict_log_proba(self, X):
148         check_is_fitted(self)
149         return self.estimator_.predict_log_proba(self.transform(X))

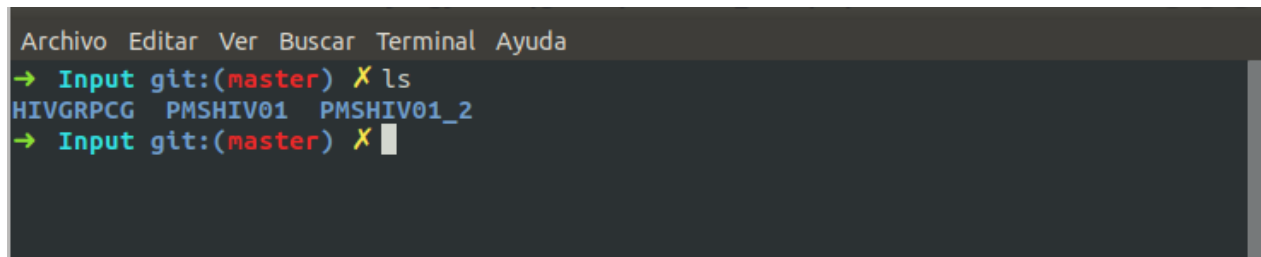
```

```

149     def _more_tags(self):
150         estimator_tags = self.estimator._get_tags()
151         return {'poor_score': True, 'allow_nan': estimator_tags.get('allow_nan', True), '
            ↳ requires_y': True,}

```

5 RESULTADOS

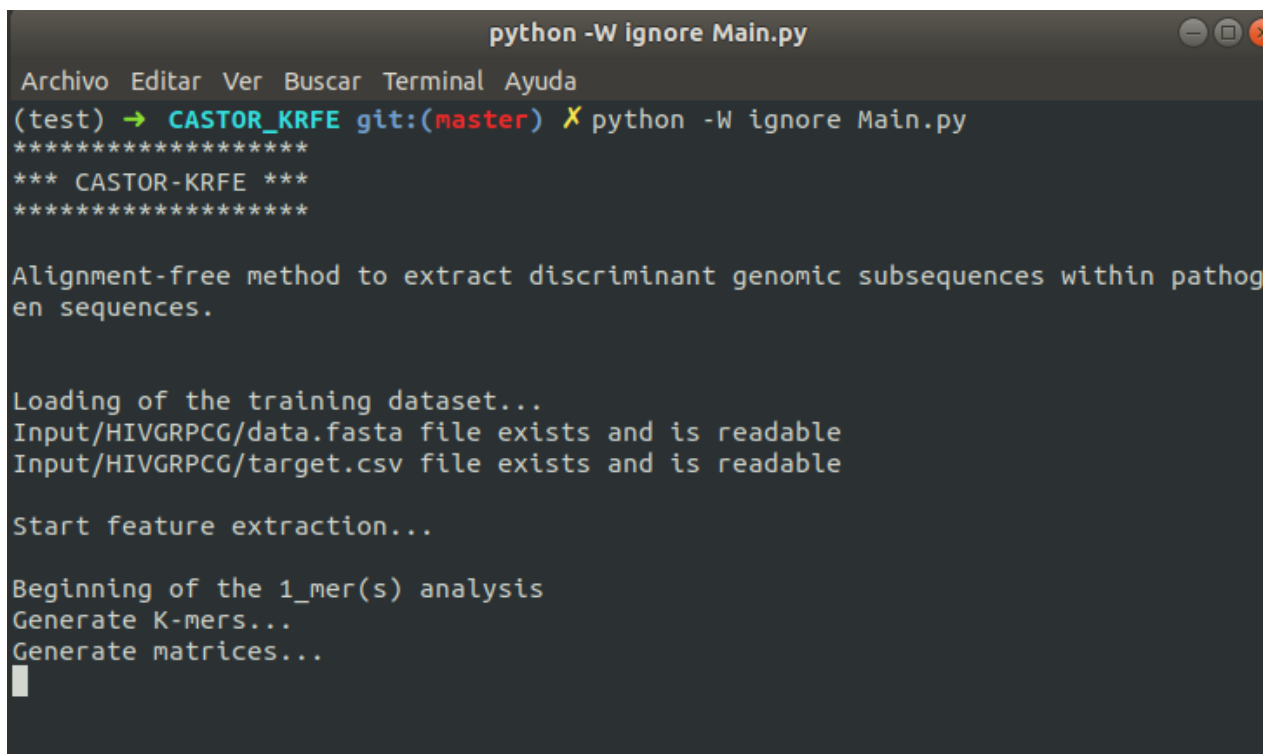


```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
→ Input git:(master) X ls
HIVGRPCG  PMSHIV01  PMSHIV01_2
→ Input git:(master) X

```

Figure 5.1: Dataset de entrada



```

python -W ignore Main.py
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
(test) → CASTOR_KRFE git:(master) X python -W ignore Main.py
*****
*** CASTOR-KRFE ***
*****

Alignment-free method to extract discriminant genomic subsequences within pathogen sequences.

Loading of the training dataset...
Input/HIVGRPCG/data.fasta file exists and is readable
Input/HIVGRPCG/target.csv file exists and is readable

Start feature extraction...

Beginning of the 1_mer(s) analysis
Generate K-mers...
Generate matrices...
█

```

Figure 5.2: Ejecutando el clasificador

```

Classification report of prediction evaluation
              precision    recall  f1-score   support

   HIV-1.M      1.000      1.000      1.000       32
   HIV-1.N      1.000      1.000      1.000       11
   HIV-1.O      1.000      1.000      1.000       32
   HIV-1.P      1.000      1.000      1.000        5

 accuracy      1.000
macro avg      1.000      1.000      1.000       80
weighted avg   1.000      1.000      1.000       80

Confusion matrix
[[32  0  0  0]
 [ 0 11  0  0]
 [ 0  0 32  0]
 [ 0  0  0  5]]

End of the program
(test) → CASTOR_KRFE git:(master) X █

```

Figure 5.3: Resultados de la clasificación

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
→ CASTOR_KRFE git:(master) X cd Output
→ Output git:(master) X ls
Analysis.png  Matrice.csv          model.pkl          Prediction_Evaluation.csv
Kmers.txt    Model_Evaluation.txt  Prediction.csv     Prediction_Evaluation.txt
→ Output git:(master) X █

```

Figure 5.4: Salida de la compilación

6 REPOSITORIO

<https://github.com/noeldiazven/Castor>

REFERENCES

- [1] D. Lebatteux, A. M. Remita, and A. B. Diallo, “Toward an alignment-free method for feature extraction and accurate classification of viral sequences,” *Journal of Computational Biology*, vol. 26, no. 6, pp. 519–535, 2019.