BTP

An Alignment-Free Scalable Feature Extraction Method for Genomic Data Clustering

Introduction

This project is all about building a model to perfom feature extraction and clustering of DNA sequence .

We have used CPF model approach for feature extraction and K-means clustering for this project purpose.

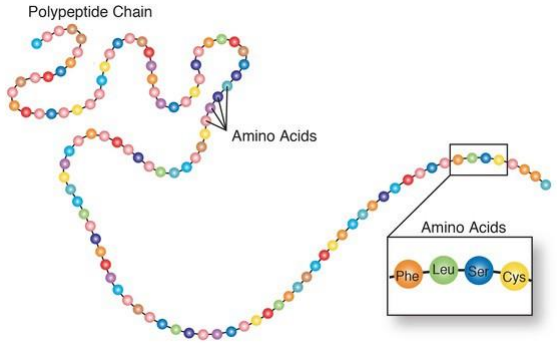
Why Feature Extraction?

We have genome data which contains nucleotides in form of characters.
The nucleotides are:
A: Adenine
T: Thymine
G: Guanine
C: Cytosine

So to apply any ML model we have to convert the data into numerical form which is done in the feature extraction stage.

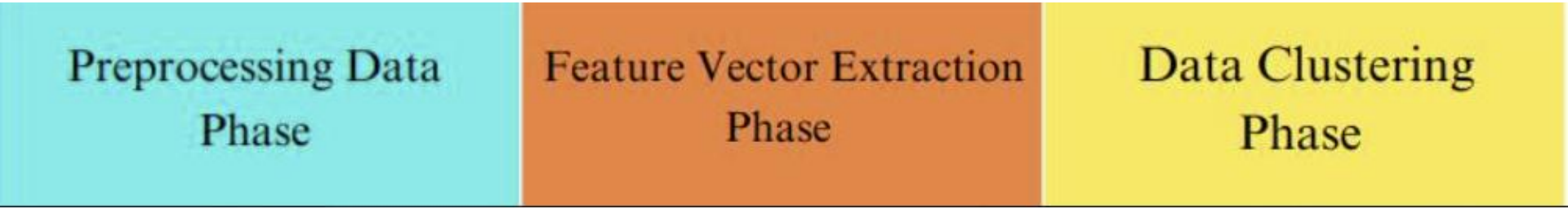
The FASTA Dataset

A FASTA file typically contains one or more sequences, with each sequence represented by a header line and a sequence of letters that correspond to the nucleotides or amino acids in the sequence. The header line of our FASTA sequence begins with a greater-than symbol (">"), followed by a description of the sequence, such as its name or accession number.



Phases

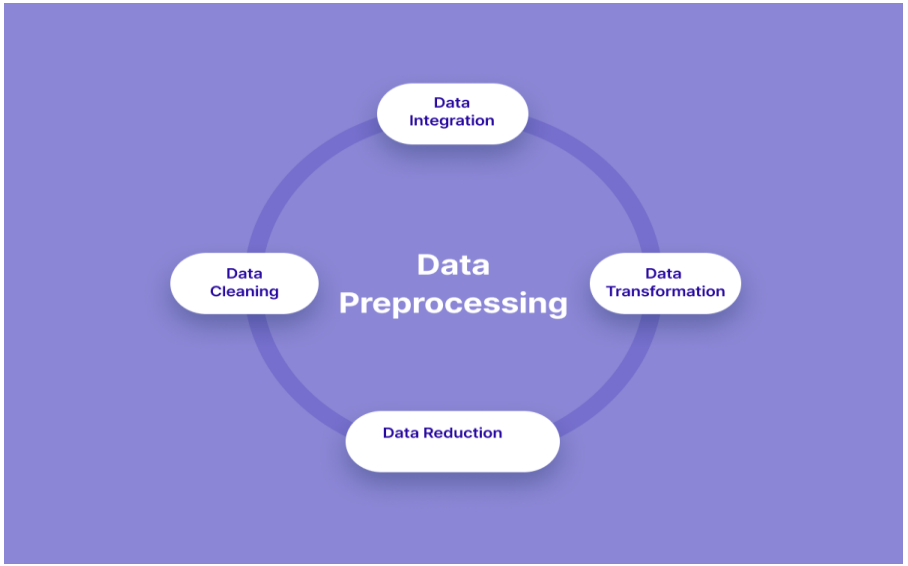
1) DATA PREPROCESSING
2) Feature Vector Extraction using Category-Position-Frequency(CPF) MODEL
3) Data Clustering using K-Means Clustering Algorithm
4) Evaluating Performance



Data Preprocessing Phase

Data Preprocessing

First step is collecting appropriate data and **preprocessing** the data which is a very important step in building the model.



Data Preprocessing Code

```
:  #Preprocessing of data with data redudancy in same file and extracting protein sequence from raw data
folder_path = "ATGC"

with open("Preprocessed_data_file/proteinSequence.txt", "w") as out_file:
    for file_name in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file_name)
            seen_sequence={}
            str_data=""
            with open(file_path, "r") as f:

                for line in f:
                    line=line.strip()
                    if len(line)>0 and line[0]=='>':
                      if len(str_data)>0:
                          if str_data not in seen_sequence:
                            seen_sequence[str_data]=True
                            if check_data_validity(str_data):
                              out_file.write(f"{str_data}\n")
                            str_data=""
                      continue
                    str_data=str_data+line

                if len(str_data)>0:
                          if str_data not in seen_sequence:
                            seen_sequence[str_data]=True
                            if check_data_validity(str_data):
                              out_file.write(f"{str_data}\n")
                            str_data=""
```

Feature Vector Extraction Phase

13D Scalable Feature Vector Extraction

The proposed 13dim-SFA extracts the features from genome sequences in terms of sequence length and entropy values.

First step: In this step we extract the length of the sequence (L).

Second step: In this step, we perform the classification and transformation of nucleotides [12] in three categories using their chemical properties.

Pyrimidine and purine class (class 1): The pyrimidine class (D) consist of two nucleotides named “C” and “T”. On the other hand, the purine class (R) consist of two nucleotides named “A” and “G”.

Keto and amino class: The keto class (E) have two nucleotide named “G” and “T” whether as the amino class (M) consist of two nucleotide “A” and “C”.

Strong hydrogen bond and weak hydrogen bond class: The strong hydrogen bond class (H) have “C” and “G”. On the other hand, weak hydrogen bond group (W) consist of the “A” and “T”.

13D Scalable Feature Vector Extraction

Third step: In this step, we find the position distribution sequence (PDS)of each word of every class. To compute the position distribution, we mark an entry of 1 for a particular word and rest will be 0.

Fourth step:- In this step, we extract another sequence made up of local frequency distribution (LFD) with the help of PDS. The local frequency can be calculated using the Eq. (1).

$$LF_i^f = \frac{1}{p_i^f - p_{i-1}^f},$$

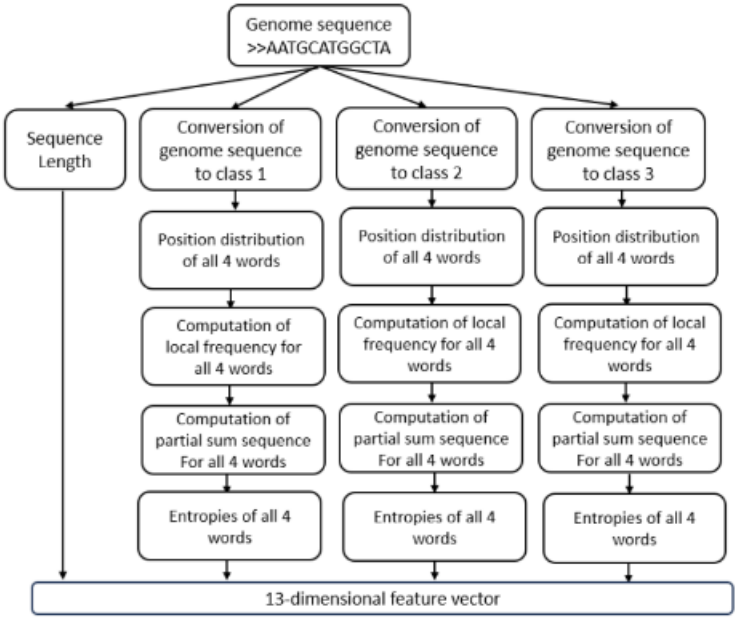
Where LF_i^f is the local frequency, p_i^f is position of the i^{th} occurrence and p_{i-1}^f denotes the position of $(i-1)^{th}$ occurrence of the word f. The value of p_0^f is considered to be 0.

Fifth step:- In this step, the partial sum sequence (PSS) is computed with the help of LFD sequence.

Sixth step:- In this step, the entropy (E) of a given PSS is computed using the below equation:

$$E = -\sum_{k=1}^{n} p_k log p_k,$$

Stages Involved



Example

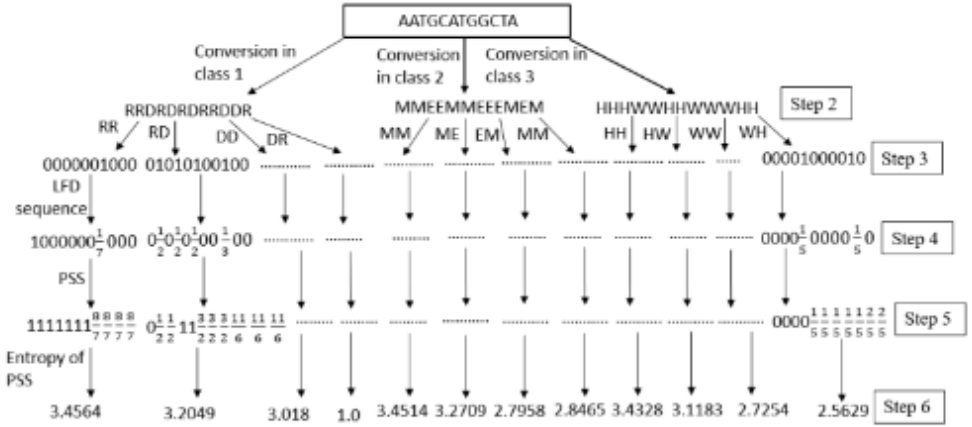


Fig. 2: Example of genome sequence

Psuedocode

Input: Genome sequence G having characters A, T, G, and C
Output: 13 dimensional feature vector

Initialize the length of the sequence (L) with 0 and a empty feature vecor.
for Each character in G **do**
    $L = L + 1$ (Increment the length of the sequence by 1)
end for
Append L to feature vector v.
for Every class (C) **do**
    $C(G) \leftarrow Genomesequence(G)$ (Transforming the G into class)
    for Every word $(w) \in C(G)$ **do**
        Create positional distribution sequence $PDS(w) \leftarrow C(G)$.
        Create local frequency distribution sequence $LFD(w) \leftarrow PDS(w)$.
        Partial sum sequence $PSS \leftarrow LFD(w)$.
        Calculate entropy of PSS using Eq. (4).
        Append E to feature vector V.
    end for
end for

Feature Vector Extraction Code

In [13]:

```
folder_path="Preprocessed_data_file"
with open('Feature_Vector/feature_vector.csv','w') as csv_file:
    writer=csv.writer(csv_file)
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        with open(file_path,"r") as f:
            for line in f:
                feature_vector=np.array([],dtype=float)
                line = line.rstrip("\n")
                #for MK
                MK_sequence=MK_group(line)
                for group in possible_groups_MK:
                    original_sequence=create_original_sequence(MK_sequence,group)
                    create_local_frequency_sequence(original_sequence)
                    create_LF_based_partial_sum_sequence(original_sequence)
                    LF_based_entropy_value=create_LF_based_entropy_value(original_sequence)
                    feature_vector=np.append(feature_vector,LF_based_entropy_value)
                #for RY
                RY_sequence=RY_group(line)
                for group in possible_groups_RY:
                    original_sequence=create_original_sequence(RY_sequence,group)
                    create_local_frequency_sequence(original_sequence)
                    create_LF_based_partial_sum_sequence(original_sequence)
                    LF_based_entropy_value=create_LF_based_entropy_value(original_sequence)
                    feature_vector=np.append(feature_vector,LF_based_entropy_value)

                #for WS
                WS_sequence=WS_group(line)
                for group in possible_groups_WS:
                    original_sequence=create_original_sequence(WS_sequence,group)
                    create_local_frequency_sequence(original_sequence)
                    create_LF_based_partial_sum_sequence(original_sequence)
                    LF_based_entropy_value=create_LF_based_entropy_value(original_sequence)
                    feature_vector=np.append(feature_vector,LF_based_entropy_value)

                writer.writerow(feature_vector)
```

Scalable Feature Extraction Using Apache Spark

Why Apache Spark?

Apache is best for scalable computing because it provides a suite of open-source software solutions that are specifically designed to handle large-scale data processing and analysis in a distributed computing environment.

- **Distributed computing**: Apache provides software solutions that enable distributed computing, which means that large-scale data processing can be divided into smaller tasks that can be executed on multiple machines in a cluster. This allows for parallel processing of data and significantly improves processing speed and scalability.
- **Fault-tolerance:** Apache's software solutions are fault-tolerant, which means that they can continue to function even if one or more nodes in the cluster fail. This is critical for large-scale data processing, where system failures are likely to occur.

- **Open-source:** Apache's software solutions are open-source, which means that they are freely available and can be customized to meet specific needs. This provides users with greater flexibility and control over their data processing and analysis workflows.

- **Community support:** Apache has a large and active community of developers and users who contribute to the development and improvement of its software solutions. This provides users with access to a wealth of knowledge and resources that can help them solve complex problems and optimize their data processing workflows.

Apache Spark in Our Project

13D Scalable Feature Vector Extraction:

- Feature Vector Extraction is made Scalable.

Apache Spark in Our Project

13D Scalable Feature Vector Extration:

- Feature Vector Extration is made Scalable.

Command to run Apache Spark in terminal

```
rd@rd-Precision-5820-Tower:~/Desktop$ cd ..
rd@rd-Precision-5820-Tower:~$ cd $SPARK_HOME
rd@rd-Precision-5820-Tower:/usr/lib/spark/spark-2.4.0-bin-hadoop2.7$ cd sbin
rd@rd-Precision-5820-Tower:/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/sbin$ ./stop-all.sh
localhost: no org.apache.spark.deploy.worker.Worker to stop
stopping org.apache.spark.deploy.master.Master
rd@rd-Precision-5820-Tower:/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/sbin$ ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /usr/lib/spark/spark-2.4.0-bin-hadoop2.7//logs/spark-rd-org.apache.spark.deploy.master.Master-1-rd-Precision-5820-Tower.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /usr/lib/spark/spark-2.4.0-bin-hadoop2.7/logs/spark-rd-org.apache.spark.deploy.worker.Worker-1-rd-Precision-5820-Tower.out
rd@rd-Precision-5820-Tower:/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/sbin$
```

Results

Metrics Used

- Silhouette index:- In unsupervised learning, silhouette index (SI) is commonly used as a measurement and analysis [10]. It depends on how similar a data point is to the other points in its cluster, known as cohesion, and to the cluster closest to it, known as separation. The value of the silhouette ranges from -1 to 1.
- Davies–Bouldin Index: The Davies-Bouldin Index (DBI) is a commonly used metric for assessing the precision of clustering results. The DBI is calculated by comparing the dissimilarity between data points within a cluster and the dissimilarity between the centroids of distinct clusters. The range of the variable's value is between zero and infinity. A low DBI indicates that clusters display a high degree of compactness and clear separation. The DBI should be minimized for optimal clustering results.

Dataset Used

1. Rice
2. Wheat

Wheat Dataset

Table 3: Results on Wheat Dataset

Count of clusters	SI			DBI		
	Proposed 13dim-SFA	17dim-FV	4dim-FV	Proposed 13dim-SFA	17dim-FV	4dim-FV
2	**0.6575**	0.3154	0.6090	**0.5422**	1.9048	0.6457
3	**0.5873**	0.2850	0.4939	**0.5347**	1.8045	0.7124
4	**0.5698**	0.2005	0.4376	**0.5361**	1.9328	0.7771
5	**0.5614**	0.1681	0.4042	**0.5390**	1.9888	0.8587
6	**0.5553**	0.1590	0.3796	**0.5117**	1.9401	0.9028
7	**0.5441**	0.1498	0.3266	**0.5106**	1.9352	0.9764
8	**0.5407**	0.1512	0.3300	**0.5075**	1.9156	1.0003
9	**0.5345**	0.1384	0.3182	**0.5123**	1.9139	1.0323
10	**0.5312**	0.1268	0.3036	**0.5199**	1.9806	1.0701

Wheat Dataset

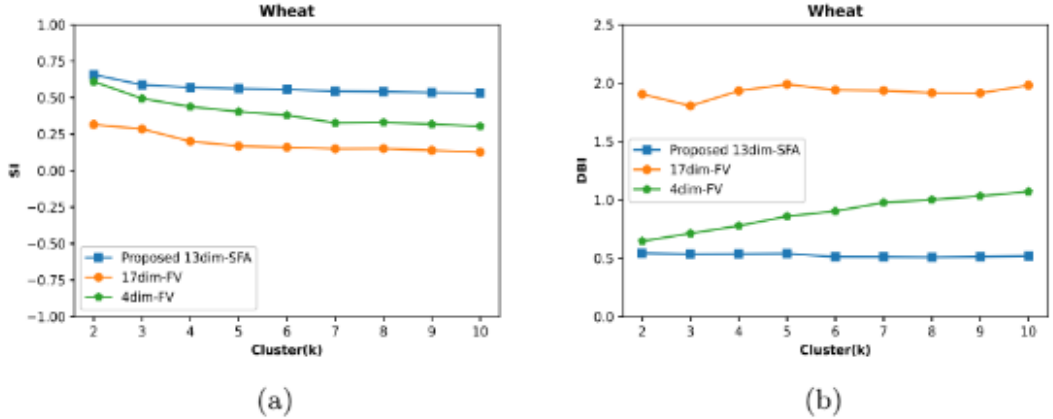


Fig: (a) SI values on wheat (b) DBI values on Wheat

Rice Dataset

Table 4: Results on Rice 1 Dataset

Count of clusters	SI			DBI		
	Proposed 13dim-SFA	17dim-FV	4dim-FV	Proposed 13dim-SFA	17dim-FV	4dim-FV
2	**0.6813**	0.4250	0.6416	**0.5614**	0.9398	0.6671
3	**0.5995**	0.2679	0.5253	**0.5434**	1.3226	0.7083
4	**0.5891**	0.2573	0.4712	**0.5107**	1.3776	0.7634
5	**0.5727**	0.1940	0.4306	**0.5137**	1.5248	0.8574
6	**0.5669**	0.1603	0.3991	**0.5193**	1.6336	0.9315
7	**0.5573**	0.1633	0.4015	**0.5154**	1.6363	0.9197
8	**0.5594**	0.1532	0.3672	**0.4941**	1.6651	0.9422
9	**0.5550**	0.1393	0.3852	**0.4904**	1.6394	0.8996
10	**0.5512**	0.1328	0.3865	**0.4930**	1.7072	0.8727

Rice Dataset

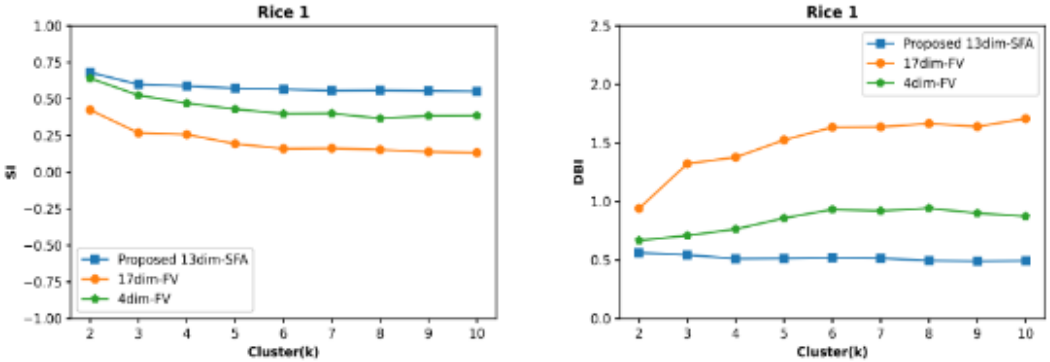


Fig: (a) SI values on wheat (b) DBI values on Wheat

Conclusion :

The experimental results demonstrate that the proposed 13dim-SFA yields a higher SI and a lower DBI in all real-world plant genome datasets of Rice and wheat, indicating that the proposed method out-performs other state-of-the-art clustering methods in terms of cluster quality.

Thank You