Feature Extraction in Short Sequence DNA Repeat Data

Dipankar Medhi
Computer Science Department
Washington State University
Dipankar.medhi@wsu.edu

Sikander Khursheed
Computer Science Department
Washington State University
Sikander.khursheed@wsu.edu

Abstract

Short sequence DNA repeats (SSRs) are relatively short (<500 base pair) patterns that recur (inexactly) in the genes of many species. Short sequence DNA repeats can be identified in all eukaryotic and many prokaryotic genomes. The dataset we that we used in our project was on SSR data for Anaplasma marginale, a pathogen which commonly infects cattle. Our primary focus was towards extracting a meaningful feature set from the SSR data for use in clustering. In the process of extraction, we incorporated various methodologies like a bag of character approach, finding n-grams, finding the first character of each DNA sequence, checking how many times a single character is repeated in the whole sequence of DNAs.

1. Introduction

Short Sequence Repeats are widely and abundantly disseminated in most nuclear eukaryotic genomes and have been widely used as molecular markers in recent years. In the data set of Short sequence repeats, we noticed that all the features were in the format of characters and getting meaningful insights from them was a challenging task. We started by extracting features from the DNA sequences, initially we applied bag of characters approach to find how many times each of the characters are repeated in each sequence, after that we searched for the length of each sequence, also counted how many times each of the characters were repeated in the whole dataset and found the first character of each sequence. After that we divided each sequence in the form of bigram, trigram and four grams to check how many repetitions of bigrams, trigrams and four grams are there in each sequence by using ngram method and library available in R. After extracting features the next step that we took was of feature selection where we removed various irrelevant features to make the data set ready for modeling. Finally, we divided the dataset into training and test set and applied Naïve Bayes classifier and found a confusion matrix also we applied a feature wrapping technique using Boruta algorithm and applied Random Forest on the feature set before and after applying Boruta algorithm and compare the confusion matrix. On comparison of both the approaches, we found out that by using the extracted features both the models return similar results.

2. Problem Definition

The patterns of the sequences were homogeneous or heterogeneous and were mostly sequences of characters. We were exploring the features to find meaningful information from within each sequence by finding the relationships of each character with every other character in the DNA sequences. Exploring these sorts of information was also important to build a regression or classification model as some of the base pairs are encoded by the amino acids. One character of a given repeat in this data set is a single amino acid. The set of all known repeats, which are one or more names and an amino acid sequence gives information about the locations where they have been reported. Also, this information can become handy in genotyping, phylogenetic characterization or pathogenic analysis.

3. Models/Algorithms /Measures

To achieve some useful results for our problem we proceeded with different measures which are stated and discussed as follow:

3.1. Feature extraction

Feature extraction is the phase in which informative patterns are extracted from the given data. Following are some feature extraction methods that we applied in our project to get

1. Bag-of approach

Bag-of approach also known as bag-of-character approach is a simplifying representation used in natural language processing and information retrieval. It helped in finding the frequency of each character in the sequence. To have a better understanding of this approach let us take a string 'adsssaggqqqe'. The possible features that we can get from bag-of-character approach are {*a=2, d=1, e=1, g=2, q=3, s=3*}.

2. Ngram approach

It is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (n − 1)order Markov model. Words are modeled such that each n-gram is composed of n words. In this approach we tried to find out features with three different variations. These variations are discussed as follow:

i. Bi-gram approach

A bigram is a ngram approach with value of n = 2. It is a sequence of two consecutive characters found in a word or sequence. To explain with an example let us take the same example explained in the bag-of approach. The possible output of bi-gram approach for the 'adsssaggqqqe' string would be {*qq = 2, ss = 2, ds = 1, sa = 1, ad = 1, gg = 1, gq = 1, qe = 1, ag = 1*}.

ii. Tri-gram approach

Tri-gram is another variation in the ngram approach with having the value of n = 3. It search for three consecutive characters in a word or sequence and gives its frequency as the output. Using the same string as an example we will get {*gqq = 1, ggq = 1, dss = 1 , ssa = 1, qqq = 1, ads = 1,sss = 1,sag = 1, agg = 1, qqe = 1*} as the output. iii. Four-gram approach

This is another approach that we performed in our project to see the outputs. Four-gram was performed by changing the value of n to 4. Taking the same example string 'adsssaggqqqe' we would have {*qqqe = 1, aggq = 1, ggqq = 1, gqqq = 1, ssag = 1, adss = 1, sssa = 1 , dsss = 1, sagg = 1*] as our output.

3. First character of each sequence

We applied this method to find if there are any common and meaningful feature that we can find by selecting the first character from the DNA sequences. To give a suitable example for this approach let us take the same string 'adsssaggqqqe' as an example. The output or the feature set that we would get for this string would be the first character of the string (I.e. *a*).

In our dataset we got seven-character values such as {*a, d, g, l, n, s, t*}. In order to use these features in our feature set we changed the character values to numeric values from 1 to 7 such as {*a = 1, d = 2, g = 3, l = 4, n = 5, s = 6, t = 7*}.

4. Counting length of each sequence

In the process of feature extraction, we thought of counting total number of characters in each sequence which may be useful in our features set. This shows us that the short sequence repeats we have lies within the range of 20-31 characters having 'adsssasgqqqessvssqlg' the smallest SSR with character count of 20 and 'adsssasgqqqessvlspsgqastssqlgvg' the longest SSR with length of 31 characters.

5. Total number of occurrences of each character

We also found repetition of each of the characters throughout the sequence by counting against the number of alphabets and found that there are characters that are only used once in the entire dataset whereas, there are few such characters that are repeated in great number. "m" was repeated only once in all the sequences and "s" was repeated 2520 times in the dataset.

6. Kmers approach

Another feature extraction approach that we followed was Kmers approach. Kmers helps to find out all the possible substrings of length K that are contained in a string. By changing the value of K, we can find out the possible substrings in a text of length K. The possible output of

Kmers approach for the string 'adsssaggqqqe' having the value of K = 2, would be {*qq = 2, ss = 2, ds = 1, sa = 1, ad = 1, gg = 1, gq = 1, qe = 1, ag = 1*}.

7. Longest common subsequence

In order to get best results, we need to gather more features and to do so we extracted some more features by finding longest common subsequence. In this approach we compared every sequence with each other and acquired the LCS for every pair. The output of this approach is the length of the longest possible subsequence. For example, we have two strings such as S1 be 'ddsssasgqqqessvssqseastssqlg' and S2 be 'adsssaggqqqessvssqsdqastssqlg' and the longest common sequence that we got from both the strings is 'dsssagqqqessvssqsastssqlg' with length of 25 characters.

At the end of the feature extraction process we grouped output from bag-of approach, bi-gram approach, tri-gram approach, four-gram approach, counting first character of each SSR, total length of each sequence and total number of occurrences of each character in one feature set and we successfully collected 1128 features of our SSR dataset.

3.2. Feature Selection

Now in order to clean the dataset and have meaningful and useful features out of those 1128 features we need to use some feature selection techniques. To select meaningful features from the dataset we used a package of R known as Caret.

This package has several functions that attempt to streamline the model building and evaluation process, as well as feature selection and other techniques. We used the following functions of this package to achieve good results:

1. Zero-and Near Zero-Variance Predictors

In our feature set that is being extracted we have many predictors which mostly have more than 80% of zero values and which might affect our final results. Zero-and near zero-variance help us to identify the predictors that at most have 0 values and remove them from the feature set.

2. Correlated Predictors

In the feature set there might be some predictors that are highly correlated to each other. These predictors may affect the results. So, in order to get accurate results, we need to remove highly correlated predictors from the feature set. To do so we have a function in R which will calculate highly correlated function. We remove the predictors that have correlation greater than 75%.

Another approach that we used in feature selection process is done using Boruta library of R. Boruta is an all relevant feature selection wrapper algorithm, capable of working with any classification method that output variable importance measure. It performs a top-down search for relevant features by comparing original attributes importance with importance achievable at

random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilize that test.

3.3. Hierarchical Clustering

As our dataset does not have class labels from which we can distinguish among the predictors. In order to have class labels, we need to make clusters of our dataset. To make clusters we used hierarchical clustering model. Before applying the clustering model to the dataset, we first need to find out the distance between the sequences and based on that distance metric we would apply hierarchical clustering model.

To find the distance we used the Levenshtein Distance Algorithm. It is a measure of the similarity between two strings, which can be referred to as the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. For example, If S is "test" and T is "test", then LD(s,t) = 0, because no transformations are needed. The strings are already identical.

After having the distance metrics, we can easily apply hierarchical clustering model and can group the data. This algorithm groups identical objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly like each other.

Now the dendrograms split in clusters can help us to label our data.

3.4. Classification Models:

Classification is the process of predicting the class of the given data. In our project we used two different kind of classification models which are briefly discussed below:

1. Naïve Bayes

Naïve Bayes is a technique to construct classifier models which can assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. Naive Bayes classifiers assume that the value of a feature is independent of the value of any other feature, given the class variable. For example, a fruit may be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

2. Random Forest

Random forests or random decision forests are learning methods for classification, regression and other tasks that operates by constructing a variety of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

4. Implementation/Analysis

To start with the implementation, we would like to discuss the dataset that we used for our project. Table 1 shows few tuples of the dataset. The dataset contains amino acid sequences and each sequence is represented by a unique name. The dataset was of a pathogen called Anaplasma marginale. This pathogen commonly infects cattle. In some cases, base pairs are replaced by the amino acids. Short-sequence repeats (SSRs) mostly occurs in both prokaryotic and eukaryotic DNA. These sequences or SSRs were extracted from different sources and were not in tidy manner but the dataset that we used in our project was collected from the course website and was in tidy and cleaned format. In this dataset we had 234 different SSRs.

Table 1

	Name	Sequence
1	A	ddsssasgqqqessvssqseastssqlg
2	B	adsssaggqqqessvssqsdqastssqlg
3	C	adsssaggqqqessvssqsgqastssqlg

Extracting features from the repeats of the SSRs was a challenging task as we needed to find meaningful features from the data set which was all characters. In order to do so we used different feature extraction approaches as discussed above.

At the initial step, we studied about different approaches that can be helpful us to extract features from the dataset. We started by applying Bag-of approach on the data set and counted the number of times each character repeated in the SSR sequences. We made twenty bags where we found that the most repeated characters were “d”, ”s”, ”a”, ”g”, ”q”, ”e”, ”v”, ”t”, ”l”. And there were eleven characters that were rarely repeated in the sequences. Figure 1 shows the output of bag-of approach.

Name	Sequence	d	s	a	g	q	e	v	t
A	ddsssasgqqqessvssqseastssqlg	2	12	2	2	5	2	1	1
B	adsssaggqqqessvssqsdqastssqlg	2	11	3	3	6	1	1	1
C	adsssaggqqqessvssqsgqastssqlg	1	11	3	4	6	1	1	1
D	adsssasgqqqessvssqseastssqlgg	1	12	3	3	5	2	1	1
E	adsssasgqqqessvssqseastssqlg	1	12	3	2	5	2	1	1

Figure 1 Bag-of approach

Moving further, we tried Ngram approach. Ngram helps to find out *n* consecutive characters in a word or string. We used three different variations by changing the value of n to 2, 3, and 4 respectively. In bigram approach we took value of *n* as 2 and were able to find 154 features. The most commonly used bigram was “ss”. By changing the value of *n* to 3 we got trigrams of the sequence. We find 370 features using trigram and the most common trigram was “ssq”. Finally, we

applied four gram using n = 4 and searched for 4 consecutive character matching in the strings and find 581 features and the most common feature in four grams was repeated only one time. N-grams can help us get meaningful information in biological contexts such as they can represent the sequences of amino acids or nucleotides[1]. For instance, the sequence "AAACG" its unigram are A,A,A,C,G. The 2-grams are AA, AA, AC, CG. Similarly, 3-grams are AAA, AAG, and ACG. Figure 2, 3, and 4 shows the features extracted from bi-gram, tri-gram and 4-gram.

Name	Sequence	ss	qq	sq	as	sg	gq
A	ddsssasgqqqessvssqseastssqlg	4	1	2	2	1	1
B	adsssaggqqqessvssqsdqastssqlg	4	1	2	1	0	1
C	adsssaggqqqessvssqsgqastssqlg	4	1	2	1	1	2
D	adsssasgqqqessvssqseastssqlgg	4	1	2	2	1	1
E	adsssasgqqqessvssqseastssqlg	4	1	2	2	1	1
F	tdsssasgqqqessvssqsgqastssqlg	4	1	2	2	2	2

Figure 2 Bi-gram approach

Name	Sequence	ssq	sgq	sss	ssa	tss
A	ddsssasgqqqessvss qseastssqlg	2	1	1	1	1
B	adsssaggqqqessvss qsdqastssqlg	2	0	1	1	1
C	adsssaggqqqessvss qsgqastssqlg	2	1	1	1	1
D	adsssasgqqqessvssqseastssqlgg	2	1	1	1	1
E	adsssasgqqqessvssqseastssqlg	2	1	1	1	1
F	tdsssasgq qqessvssqsgqastss qlg	2	2	1	1	1

Figure 3 Tri-gram approach

Name	Sequence	sssa	tssq	stss	asts	dsss
A	ddsssasgqqqessvssqseastssqlg	1	1	1	1	1
B	adsssaggqqqessvssqsdqastssqlg	1	1	1	1	1
C	adsssaggqqqessvssqsgqastssqlg	1	1	1	1	1
D	adsssasgqqqessvssqseastssqlgg	1	1	1	1	1
E	adsssasgqqqessvssqseastssqlg	1	1	1	1	1
F	tdsssasgqqqessvssqsgqastssqlg	1	1	1	1	1

Figure 4 4-gram approach

On comparing the results of bag-of approach with Ngram and Kmers approach find that bag-of approach only provides the frequency counts of the characters it was not following any order. However, N- gram helps in parsing the sequences and describe how well the language model predicts a new text composed of unseen sentences.

Our main focus was to extract as many features as possible. In order to do that, we took another approach known as Kmers. It also helps to find *K* consecutive characters in a string and return its frequency. In Kmers we also used 3 different variations by picking k as 2, 3, and 4. Following figures shows the result features of all the approaches.

Name	Sequence	ss	qq	sq	as	sg	gq
A	ddsssasgqqqessvssqseastssqlg	4	1	2	2	1	1
B	adsssaggqqqessvssqsdqastssqlg	4	1	2	1	0	1
C	adsssaggqqqessvssqsgqastssqlg	4	1	2	1	1	2
D	adsssasgqqqessvssqseastssqlgg	4	1	2	2	1	1
E	adsssasgqqqessvssqseastssqlg	4	1	2	2	1	1
F	tdsssasgqqqessvssqsgqastssqlg	4	1	2	2	2	2

Figure 5 2-mers approach

Name	Sequence	ssq	sgq	sss	ssa	tss
A	ddsssasgqqqessvssqseastssqlg	2	1	1	1	1
B	adsssaggqqqessvssqsdqastssqlg	2	0	1	1	1
C	adsssaggqqqessvssqsgqastssqlg	2	1	1	1	1
D	adsssasgqqqessvssqseastssqlgg	2	1	1	1	1
E	adsssasgqqqessvssqseastssqlg	2	1	1	1	1
F	tdsssasgqqqessvssqsgqastssqlg	2	2	1	1	1

Figure 6 3-mers approach

Name	Sequence	sssa	tssq	stss	asts	dsss
A	ddsssasgqqqessvssqseastssqlg	1	1	1	1	1
B	adsssaggqqqessvssqsdqastssqlg	1	1	1	1	1
C	adsssaggqqqessvssqsgqastssqlg	1	1	1	1	1
D	adsssasgqqqessvssqseastssqlgg	1	1	1	1	1
E	adsssasgqqqessvssqseastssqlg	1	1	1	1	1
F	tdsssasgqqqessvssqsgqastssqlg	1	1	1	1	1

Figure 74-mers approach

One thing we find out in this approach is that by using Kmers approach we would get the same results as that of n-gram approach. So, to avoid data duplication in our feature set we ignored output of one of the approaches.

After checking for character repetitions, we also tried to find out some other features that may be useful for our final dataset. We gathered some more features by using three different measures. The first one was to count the total length of each sequence, which gave us integer value between the ranges of 20 to 31. This means that the SSRs that we have in our dataset are neither smaller than 20 characters and nor longer than 31 characters. The other two approaches we carried out in order to find out some meaningful information were counting the first character of each sequence and counting each individual character or amino acid in the whole dataset of 234 sequences.

To achieve best results, we need to gather more features. We took one more approach known as Longest Common Subsequence (LCS). In this approach each SSR is compared with another sequence and tried to find out the length of the longest subsequence and its quality similarity

index. Figure 8 shows the features extracted from longest common subsequence method. We can clearly see in the figure that we compared two strings and find longest subsequence possible from both the strings.

S1	S2	LCS	LengthLCS	QualitySimilarityIndex
ddsssasgqqqessvssqseastssqlg	ddsssasgqqqessvssqseastssqlg	ddsssasgqqqessvssqseastssqlg	28	1.0000000
ddsssasgqqqessvssqseastssqlg	adsssaggqqqessvssqsdqastssqlg	dsssagqqqessvssqsastssqlg	25	0.8620690
ddsssasgqqqessvssqseastssqlg	adsssaggqqqessvssqsgqastssqlg	dsssagqqqessvssqsastssqlg	25	0.8620690
ddsssasgqqqessvssqseastssqlg	adsssasgqqqessvssqseastssqlgg	dsssasgqqqessvssqseastssqlg	27	0.9310345
ddsssasgqqqessvssqseastssqlg	adsssasgqqqessvssqseastssqlg	dsssasgqqqessvssqseastssqlg	27	0.9642857
ddsssasgqqqessvssqseastssqlg	tdsssasgqqqessvssqsgqastssqlg	dsssasgqqqessvssqsastssqlg	26	0.8965517

Figure 8 Longest Common Subsequence

By merging all the outputs from different approaches, we took we were able to collect around 1128 features. After collecting features, now the next phase that we worked on was to collect meaningful features and reduce our feature set so that we can build a successful regression or classification model. In order to do so we followed two different approaches. The first one was by applying caret package available in R using which we removed near zero value feature sets and highly correlated values. These two approaches helped us a lot to shrink our feature set. Removing zero-variance reduced the feature set from 1128 features to 184 features by removing all the features that had mostly 0 as an output in other words the feature set that had 80% or more 0 values were removed. By reducing the feature set we still had some highly correlated predictors that may deviate the final result. So, by finding highly correlated predictors we find out that we still have 122 highly correlated predictors and we removed those 122 predictors and were left with 62 predictors which means that we have removed large number of irrelevant data.

Another approach that we took for feature selection was using Boruta package. It is a wrapper algorithm and it reduce the dataset by identifying meaningful and not so important features. Function in Boruta package are used to find features and its importance in the feature set. It performs a top-down search for relevant features by comparing original attributes importance with importance achievable at random and highlighting irrelevant features to get stable end results. Using Boruta algorithm on the feature set we found various shadow attributes, tentative attributes, confirmed and unconfirmed attributes in the feature set.

The next step that we were trying to perform on our feature set was to apply classification model and predict results for our testing set. But before we proceed with that we analyzed our feature set and came to know that we didn't had any class label from which we can use as a predictor for the feature set. We tried to make clusters of our SSRs using hierarchical clustering model. We tried to find out distance metrics using different distance formulas, and we were successful in finding it by using Levenshtein distance formula. Levenshtein is a distance finding formula it finds distance by finding the similarity between two strings. The distance is the number of deletions, insertions, or substitutions required to transform the source word to target word. By applying agglomerative hierarchical clustering algorithm on the Levenshtein distances and we get the dendrograms and its split in cluster can help us to label the data from 1 to 6 as shown in figure 9.

After creating the class, we randomly divided the feature set into training and testing set to use for classification models by using 75:25 rule.

We used two different kind of classification models to predict results and compare those results. The first approach we took was Naïve Bayes classification. Naïve Bayes is a method used to construct classifier models which can assign class labels to problem instances. The next classification approach that we followed was using Random Forest. It is a learning method for classification, regression and other tasks that operates by constructing a variety of decision trees at training time and giving output as the class that is the mode of the classes of the individual trees.

We compared our results by applying both the models and noticed that the accuracy of the model remains similar in both Naïve Bayes and Random forest. Whereas, applying random forest on the feature set before and after applying Boruta algorithm, we find improvement in accuracy of 2%.

5. Results and Discussion

Based on our implementation and measures we took to perform operations on our dataset and we got different results which we would like to discuss over here.

We extracted features using different approaches. On comparison of the results of bag-of approach with Ngram and Kmers approach we found out that bag-of approach only provides the frequency counts of the characters it was not following any order. As we can clearly see in figure 1 that bag-of approach returns only the count of each character in the sequences. For example, in the first sequence character *s* is repeated 12 times and character d is repeated 2 times and so on. Whereas, n-gram and Kmers tries to find pattern of characters following order. In other words, it works by matching characters in sequence based on the window of K or n.

From the features that we extracted using n-gram and Kmers approaches we found out that by using Kmers approach we would get the same results as that of n-gram approach, as shown in figures (3,4,5 and 6,7,8). So, rather than having same features from different approaches we included only the output of n-gram approach so that we can avoid data duplication.

Another feature extraction method we used was finding longest common subsequence. Finding LCS is time consuming process and the complexity of this approach is quiet high[2]. In order to avoid time complexity of our program we decided not to include LCS in our final feature set.

From feature extraction process we were able to extract more than 1000 features, which contained irrelevant features, which may impact the final results. In order to get meaningful and useful features we removed near-zero variance predictors as well as highly correlated predictors. The reason behind using these functions were to extract useful features and get rid of untidy data.

Our dataset does not contain any class labels from which the data can be distinguishable or classified. In order to have some class labels for the dataset we used Levenshtein distance metrics which helped us to make clusters of the data sequences. Based on those cluster we grouped the data as shown in figure 9. In the figure 9 the dendrograms split in clusters can help us to label our data.

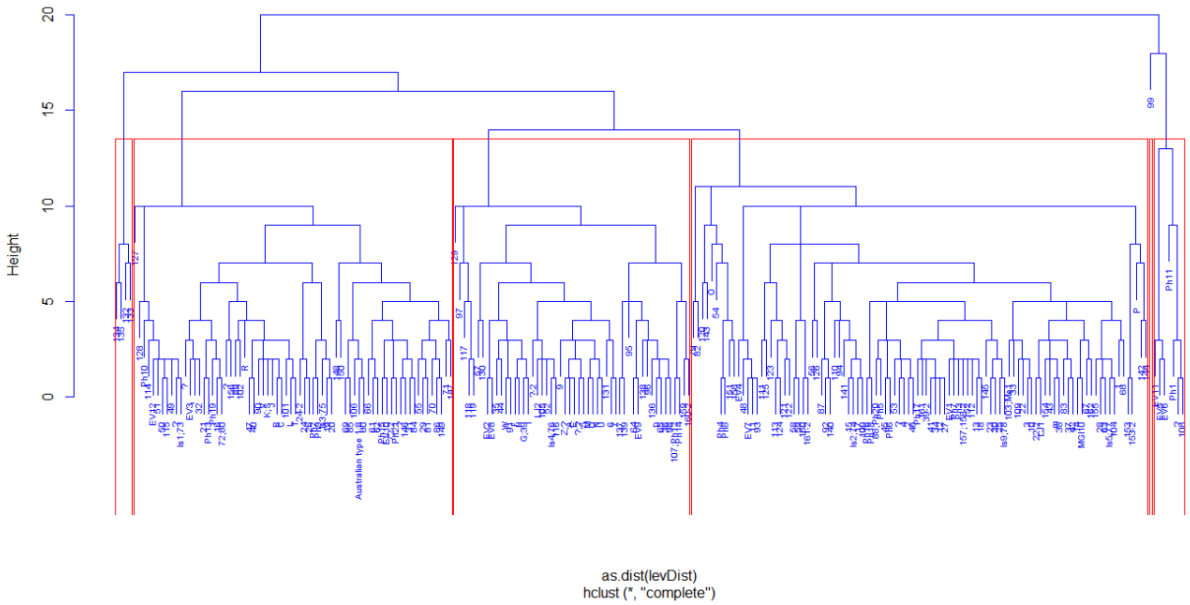

Figure 9 Hierarchical Clustering

After reducing feature set, we split the data into training and testing based on 75:25 ratio and applied Naïve Bayes classification model on the feature set to classify the models. Figure 10 shows the confusion matrix of Naïve Bayes Classifier.

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4  5  6
         1 13  0  0  0  0  0
         2  0 20  1  0  0  0
         3  0  0 24  0  0  0
         4  0  0  0  0  0  0
         5  0  0  0  0  0  0
         6  0  0  0  0  0  1

Overall Statistics

               Accuracy : 0.9831
                 95% CI : (0.9091, 0.9996)
    No Information Rate : 0.4237
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9742
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            1.0000   1.0000   0.9600       NA       NA  1.00000
Specificity            1.0000   0.9744   1.0000        1        1  1.00000
Pos Pred Value         1.0000   0.9524   1.0000       NA       NA  1.00000
Neg Pred Value         1.0000   1.0000   0.9714       NA       NA  1.00000
Prevalence             0.2203   0.3390   0.4237        0        0  0.01695
Detection Rate         0.2203   0.3390   0.4068        0        0  0.01695
Detection Prevalence   0.2203   0.3559   0.4068        0        0  0.01695
Balanced Accuracy      1.0000   0.9872   0.9800       NA       NA  1.00000
```

Figure 10 Confusion matrix of Naive Bayes Classification Model

To compare our model against Naïve Bayes model we applied a feature wrapping algorithm, Boruta, and segregated the features into shadow attributes, tentative attributes, confirmed and unconfirmed attributes in the feature set as shown in figure 11. Whereas, figure 12 represent the density of the segregated features obtained from the Boruta algorithm.

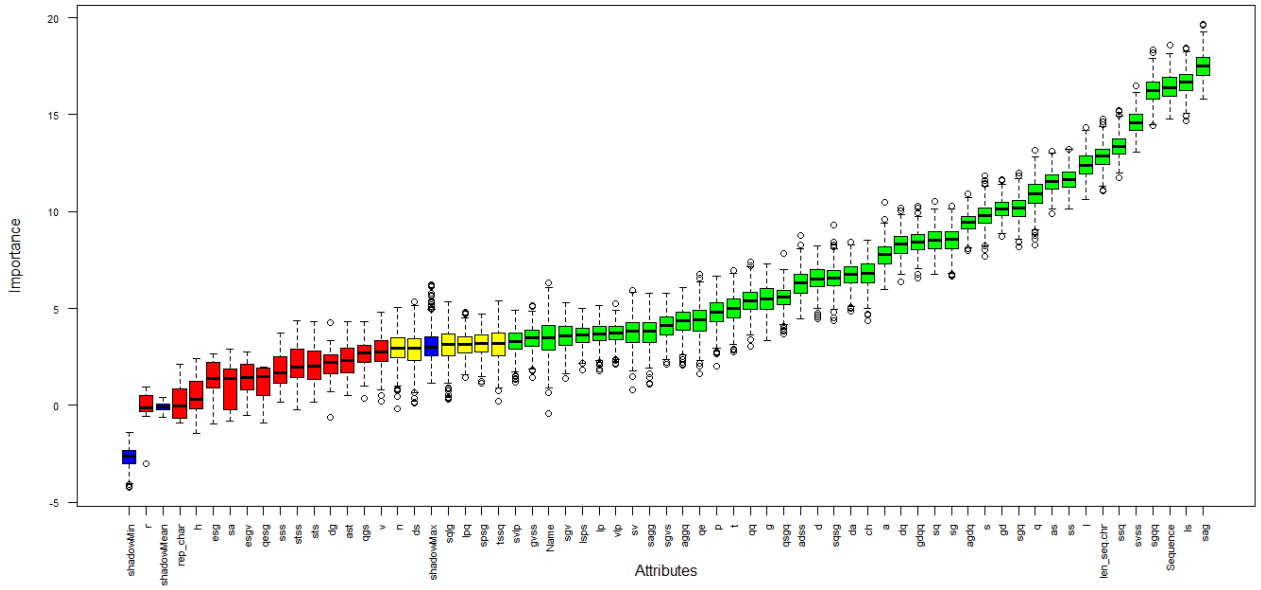

Figure 11 Boruta plot

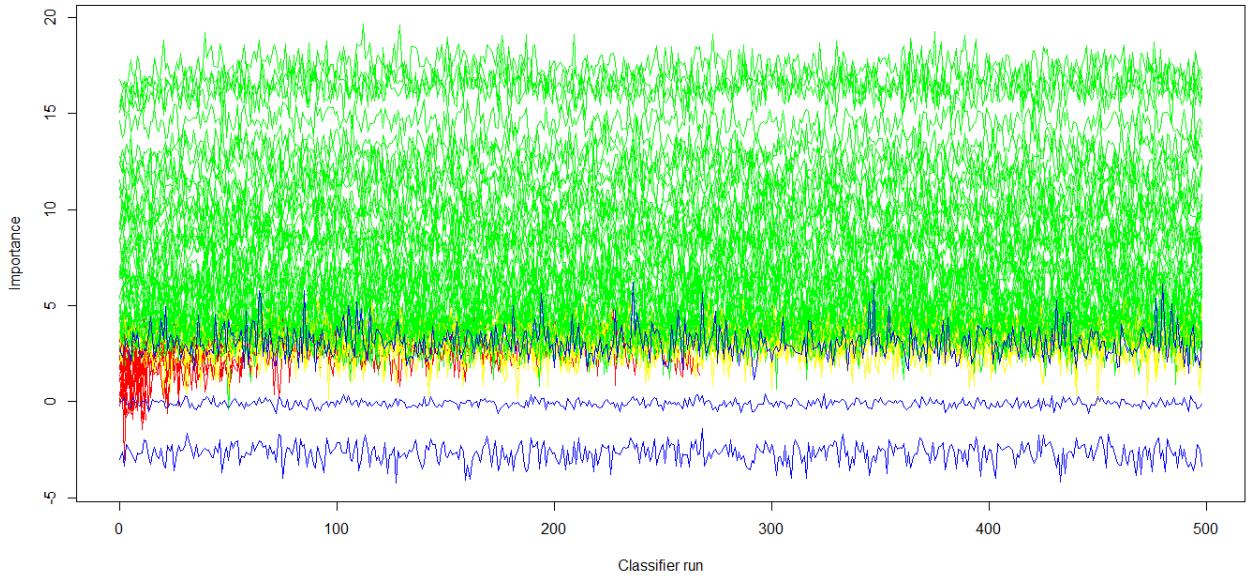

Figure 12 Density of feature set

We applied another classification model named random forest to compare the results with Naïve Bayes. We applied random forest on the original feature set as well as on the confirmed feature set that was obtained by Boruta algorithm. Confusion metrics of both the approaches are shown in the figure 13 & 14.

```
> confusionMatrix(p0,test$Class)
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4  5  6
         1 12  1  0  0  0  0
         2  1 20  0  0  0  0
         3  1  1 30  0  0  1
         4  0  0  0  2  0  0
         5  0  0  0  0  0  0
         6  0  0  0  0  0  0

Overall Statistics

               Accuracy : 0.9275
                 95% CI : (0.8389, 0.9761)
    No Information Rate : 0.4348
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8895
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            0.8571   0.9091   1.0000  1.00000       NA  0.00000
Specificity            0.9818   0.9787   0.9231  1.00000        1  1.00000
Pos Pred Value         0.9231   0.9524   0.9091  1.00000       NA      NaN
Neg Pred Value         0.9643   0.9583   1.0000  1.00000       NA  0.98551
Prevalence             0.2029   0.3188   0.4348  0.02899        0  0.01449
Detection Rate         0.1739   0.2899   0.4348  0.02899        0  0.00000
Detection Prevalence   0.1884   0.3043   0.4783  0.02899        0  0.00000
Balanced Accuracy      0.9195   0.9439   0.9615  1.00000       NA  0.50000
```

Figure 13 Confusion Matrix of Random Forest Algorithm

```
> confusionMatrix(p1,test$Class)
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  4  5  6
         1 12  1  0  0  0  0
         2  1 20  0  0  0  0
         3  1  1 30  0  0  0
         4  0  0  0  2  0  0
         5  0  0  0  0  0  0
         6  0  0  0  0  0  1

Overall Statistics

               Accuracy : 0.942
                 95% CI : (0.8582, 0.984)
    No Information Rate : 0.4348
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9124
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
Sensitivity            0.8571   0.9091   1.0000  1.00000       NA  1.00000
Specificity            0.9818   0.9787   0.9487  1.00000        1  1.00000
Pos Pred Value         0.9231   0.9524   0.9375  1.00000       NA  1.00000
Neg Pred Value         0.9643   0.9583   1.0000  1.00000       NA  1.00000
Prevalence             0.2029   0.3188   0.4348  0.02899        0  0.01449
Detection Rate         0.1739   0.2899   0.4348  0.02899        0  0.01449
Detection Prevalence   0.1884   0.3043   0.4638  0.02899        0  0.01449
Balanced Accuracy      0.9195   0.9439   0.9744  1.00000       NA  1.00000
```

Figure 14 Confusion Matrix of Random Forest Algorithm for confirmed attributes

From the results generated by applying both the approaches we found that Naïve Bayes classification model provides accuracy of ~98% whereas from Random Forest classification we obtained accuracy of ~92% before applying Boruta algorithm and ~94% after applying Boruta algorithm. Which shows that the features that we extracted are useful for any regression, classification or clustering model.

6. Related works

Measures that we used in our project can be used to investigate various genotyping results in DNAs. Its usage are prominently seen in building repeat analyzers for SSR's[3].We can also see its usage in building classification model by finding interesting features from DNA Methylation Sequencing Data for various cancer patterns[4]. Also comparative analysis of protein sequences can be achieved by extracting features similar way[5].

7. Conclusion

In summary, we started with data set of sequences of SSR's of Anaplasma Marginale and have extracted various features. In order to select meaningful features from the extracted feature set by applying various feature extraction methods like removing zeros from the feature set and removing highly correlated values. We calculated distance of the sequences which were useful in classifying the data. A feature wrapping technique was used to find the meaningful features. Finally, we checked our features by applying them into two classification models Naïve Bayes and Random forest and found their confusion metrics. In future we can use these methods to extract various meaningful features from DNA sequences of patients suffering from different kind of diseases to extract meaningful patterns from their DNAs to know the causes for such diseases also we can foresee a great application of such feature extraction methods in the field of Bioinformatics.

Bibliography

[1] W. Liang and Z. KaiYong, "Segmenting DNA sequence into 'words,'" p. 12.

[2] A. Apostolico and C. Guerra, "The longest common subsequence problem revisited," *Algorithmica*, vol. 2, no. 1–4, pp. 315–336, Nov. 1987.

[3] H. N. Catanese, K. A. Brayton, and A. H. Gebremedhin, "RepeatAnalyzer: a tool for analysing and managing short-sequence repeat data," *BMC Genomics*, vol. 17, no. 1, Dec. 2016.

[4] Z. Yang, M. Jin, Z. Zhang, J. Lu, and K. Hao, "Classification Based on Feature Extraction For Hepatocellular Carcinoma Diagnosis Using High-throughput Dna Methylation Sequencing Data," *Procedia Comput. Sci.*, vol. 107, pp. 412–417, 2017.

[5] M. Ganapathiraju, D. Weisser, R. Rosenfeld, J. Carbonell, R. Reddy, and J. Klein-Seetharaman, "Comparative n-gram analysis of whole-genome protein sequences," in *Proceedings of the second international conference on Human Language Technology Research* -, San Diego, California, 2002, p. 76.

[6] X. Ma and L. Hu, "Extracting Sequence Features to Predict DNA-Binding Proteins Using Support Vector Machine," in *2013 International Conference on Computational and Information Sciences*, Shiyang, China, 2013, pp. 152–155.

[7] L. Wang, "Random Forests for Prediction of DNA-Binding Residues in Protein Sequences Using Evolutionary Information," in *2008 Second International Conference on Future Generation Communication and Networking*, Hainan, China, 2008, pp. 24–29.

Link to code:

https://drive.google.com/file/d/1bib_0qGC4f1Ey-nE65-JKy_59MKy9Knf/view?usp=sharing