International journal of advances in engineering and management (IJAEM)
Volume 3, issue 6 June 2021, pp: 1759-1767 www.ijaem.net ISSN: 2395-5252

Genome Sequencing and Classifier

[1,2]Puneet Gupta, Gaurav Melkani, Dr Sunil Maggu, Dr Anu Rathee
Maharaja Agrasen Institute of Technology Delhi
(Assistant professor, Department of Computer Science and Engineering): Maharaja Agrasen Institute of Technology Delhi

Submitted: 01-06-2021 Revised: 14-06-2021 Accepted: 16-06-2021

ABSTRACT: This research paper is about to perform sequence and classification on DNA Sequence using machine learning.
Genomics is a branch of molecular biology focused on studying all aspects of a genome, or the complete set of genes within a particular organism. Today, machine learning is playing an integral role in the evolution of the field of genomics. The ability to sequence DNA provides researchers with the ability to “read” the genetic blueprint that directs all the activities of a living organism. In the ML categories of classification, clustering and regression have been proven useful for solving biological research questions such as gene signatures, functional genomics, gene-phenotype associations and gene interactions.

KEYWORDS: DNA Sequence,Python K-Mer,Classification,Count vectorizer, NLP, Naive Bayes,Machine Learning,Bag of Words.

I. INTRODUCTION

In this paper we have done DNA sequencing and classification on Human, Chimpanzee and dogs genes respectively.. To provide context, the central dogma of biology is summarized as the pathway from DNA to RNA to Protein. DNA is composed of base pairs, based on 4 basic units (A, C, G and T) called nucleotides: A pairs with T, and C pairs with G. DNA is organized into chromosomes and humans have a total of 23 pairs.
With the massive generation of data, the era known as ‘big’ data, deep learning (DL) approaches appeared as a discipline of ML that are considered to be more efficient and effective when we deal with big amounts of data. DL has also proven to provide models with higher accuracy that are efficient at discovering patterns in high-dimensional data making them applicable to a variety of domains.

The complexity and escalation of the NGS data pose problems; sharing, storing, archiving, and analyzing data as large as 1 TB per sample is an issue. The current sequencing platforms are capable of producing 13 quadrillion DNA bases limit of **NGS technologies** is evaluated to be 13 quadrillion DNA bases per year and is hard to manage.

However, this limitation is overcome by the development of many **machine learning** algorithms, NGS software, and big data analytics. Big data analytics, a new trend in research, promises the development of significant approaches for the analysis of complex NGS data using customized next-generation sequencing software.

Both machine learning and **data science** (like deep learning) are emerging as the latest and the most efficient approaches to speed up the sequencing and analysis process. The development of multiple algorithms like indexes, hash tables, and spaced-seed has led to the optimization of the NGS data analysis.

[1].Treating DNA as a language known as k-mers. In bioinformatics, **k-mers** are subsequences of length contained within a biological sequence. Primarily used within the context of computational genomics and sequence analysis, in which k-mers are composed of nucleotides (i.e. A, T, G, and C), k-mers are capitalized upon to assemble DNA sequences. A method of visualizing k-mers, the k-mer spectrum, shows the multiplicity of each k-mer in a sequence versus the number of k-mers with that multiplicity. The frequency of k-mer usage is affected by numerous forces, working at multiple levels, which are often in conflict. It is important to note that k-mers for higher values of k are affected by the forces affecting lower values of k as well. For example, if the 1-mer A does not occur in a sequence, none of the 2-mers containing A (AA, AT, AG, and AC) will occur either, thereby linking the effects of the different forces.

[2]. Natural language processing is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human languages, in particular how to program computers to process and analyze large amounts of

natural language data.The internal combustion engine is a device which basically converts the heat energy into mechanical energy. The cam has been an integral part of the IC engine from its invention. As with the demands for better fuel economy, more power, and less pollution, motor engineers around the world are pursuing a radical “camless” design that promises to deliver the internal combustion engine’s biggest efficiency improvement in years. The article looks at the working of the electrohydraulic camless engine, its general features and benefits over conventional engines. In this article we focused on a basic overview of camless engine along with its design principle, components and its merits over other conventional engines.

[3]. CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors. CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

Inside CountVectorizer, these words are not stored as strings**.** We will be creating vectors that have a dimensionality equal to the size of our vocabulary, and if the text data features that vocabulary word, we will put a one in that dimension. Every time we encounter that word again, we will increase the count, leaving 0s everywhere we did not find the word even once.

[4]. We need to now convert the lists of k-mers for each gene into string sentences of words that can be used to create the Bag of Words model.
[5]. Creating the Bag of Words model using CountVectorizer(). This is equivalent to k-mer counting. Converted our k-mer words into uniform length numerical vectors that represent counts for every k-mer in the vocabulary. So, for humans we have 4380 genes converted into uniform length feature vectors of 4-gram k-mer (length 6) counts. For chimp and dog, we have the same number of features with 1682 and 820 genes respectively.

II. LIETERATURE REVIEW

Algorithms that have been used for protein sequence classification can be classified roughly into several types, depending on whether they are based on the K-Nearest Neighbor (K-NN) approach, the Hidden Markov Model (HMM) approach, or the Support Vector Machine (SVM) approach or any other classification algorithms. In the context of protein sequence classification, Markov models (MMs) are used to capture dependencies between the neighboring sequence elements. MMs are among the most widely used generative models of sequence data [4]. In a kth order MM, the sequence elements satisfy the Markov property: each element is independent of the rest given the K preceding elements. Begleiter et al.[2] have examined methods for prediction using variable order MMs, including probabilistic suffix trees, which can be viewed as variants of abstraction wherein the abstractions are constrained to share suffixes.

Due to its simplicity, the K-NN approach to classification (Fix et al., 1949) [8] has been popular in the biological domain (Deshpande et al., 2002; Lu et al., 2003) [4, 11]. Given a database of pre-classified sequences, a new sequence can be classified by finding k sequences in the database. It is then assigned to the class that the majority of these k sequences belong to. The key step in building a K-NN classifier is to determine how similar two sequences are, and the measure of similarity is usually determined by computing global or local alignment scores. For this purpose, the most popular algorithm used is the Smith-Waterman dynamic programming algorithm (Smith et al., 1981) [15]. This algorithm is relatively accurate, but it is not very computationally efficient. Heuristic algorithms such as BLAST (Altschul et al., 1990) [1] and FASTA (Pearson, 1990) [13] have therefore been developed to trade reduced accuracy for improved efficiency.

The HMM-based approaches (Rabiner, 1989) [14] to protein sequence classification have been shown to be effective in detecting for conserved residue patterns in a set of protein sequences (Eddy et al., 1995; Hughey et al., 1996 ;) [6], [9]. A typical HMM consists of a chain of match, insert, and delete states in a Markov chain, with all transitions between states and all residue costs in the insert and match states trained to specific probabilities. When the HMM is trained on a set of sequences that are members of a given protein family, the model parameters are learned via an expectation-maximization approach and a form of dynamic programming is used to detect for similarity. The resulting HMM can identify the positions of residues that can describe conserved primary structures of a family and it can then be used to discriminate between family and non-family members.

The SVM-based approaches (Cristianini et al., 2002; Vapnik, 1998) [3] to classification use both positive and negative examples when training a classifier. They perform protein sequence classification by mapping the input training sequences into a high dimensional feature space and try to locate in the feature space a plane that maintains a maximum margin from any point in the training set. Then the class label of the unclassified sequence is predicted by mapping it into the feature space and deciding on which side of the separating plane, the given sequence lies. The SVM-pairwise method (Schölkopf et al., 2004) [16] also requires that a given set of protein sequences be converted into fixed-length vectors first. SVM is then trained from the vectorized protein sequences. A list of pairwise sequence similarity scores are computed by using the dynamic programming algorithm.

The K-NN-, HMM-, and SVM-based algorithms are most commonly used for protein sequence classification. In K-NN-based approach, the number of k needs to be determined in advance. Also, pairwise alignment is computationally inefficient and the reliability of similarity detection falls rapidly whenever the pairwise sequence identity drops below 30%. In HMM based algorithms, many parameters are needed to be estimated accurately and this requires a large amount of training data which may not always be readily available. For SVM based approaches, all input sequences need to be aligned beforehand in order to transform them into fixed-length vectors in the feature space and the alignment process can be difficult and time-consuming.

III. PROBLEM OBJECTIVE AND METHODOLOGY

How can we verify and Identify particular class of the Gene ?

In this Research our objective is to build a classification model that is trained on the human DNA sequence and can predict a gene family based on the DNA sequence of the coding sequence. To test the model, we will use the DNA sequence of humans, dogs, and chimpanzees and compare the accuracies perform sequencing and classification. Sequencing DNA means determining the order of the four chemical building blocks - called "bases" - that make up the DNA molecule. The sequence tells scientists the kind of genetic information that is carried in a particular DNA segment. Eventually we will do classification. In classification, the model produced assign input to one of the classes depending on the decision rules. As a data-driven science, genomics extensively utilizes machine learning to capture dependencies in data and infer new biological hypotheses. Nonetheless, the ability to extract new insights from the exponentially increasing volume of genomics data requires more powerful machine learning models. By efficiently leveraging large data sets, deep learning has reconstructed fields such as computer vision and The double-helix is the correct chemical representation of DNA. But DNA is special. It's a nucleotide made of four types of nitrogen bases: Adenine (A), Thymine (T), Guanine (G), and Cytosine. We always call them A, C, Gand T.

DNA sequencing is the process of determining the nucleic acid sequence – the order of nucleotides in DNA. It includes any method or technology that is used to determine the order of the four bases: adenine, guanine, cytosine, and thymine.

Sequencing DNA means determining the order of the four chemical building blocks - called "bases" - that make up the DNA molecule.

The sequence tells scientists the kind of genetic information that is carried in a particular DNA segment.

For example, scientists can use sequence information to determine which stretches of DNA contain genes and which stretches carry regulatory instructions, turning genes on or off.

method of preference for many genomics modeling tasks including predicting the influence of genetic on gene regulatory mechanisms such as DNA receptiveness and splicing.

DNA and protein sequences can be seen as the language of life. The language encodes instructions as well as functions for the molecules that are found in all life forms. The sequence language resemblance continues with the genome as the book, subsequences (genes and gene families) are sentences and chapters, k-mers and peptides are words, and nucleotide bases and amino acids are the alphabets. Since the relationship seems so likely, it stands to reason that the natural language processing(NLP) should also implement the natural language of DNA and protein sequences. The method we use here is manageable and easy. We first take the long biological sequence and break it down into k-mer length overlapping "words". For example, if we use "words" of length 6 (hexamers), "ATGCATGCA" becomes: 'ATGCAT', 'TGCATG', 'GCATGC', 'CATGCA'. Hence our example sequence is broken down into 4 hexamer words.

In genomics, we refer to these types of manipulations as "k-mer counting", or counting the occurrences of each possible k-mer sequence and Python natural language processing tools make it easy.

SOURCE CODE FOR DNA SEQUENCING AND CLASSIFICATION

```
def Kmers_funct(seq, size):
    return [seq[x:x+size].lower() for x in range(len(seq) - size + 1)]


#So let’s try it out with a simple sequence:
mySeq = 'GTGCCCAGGTTCAGTGAGTGACACAGGCAG'
Kmers_funct(mySeq, size=7)
words = Kmers_funct(mySeq, size=6)
joined_sentence = ' '.join(words)
joined_sentence
mySeq1 = 'TCTCACACATGTGCCAATCACTGTCACCC'
mySeq2 = 'GTGCCCAGGTTCAGTGAGTGACACAGGCAG'
sentence1 = ' '.join(Kmers_funct(mySeq1, size=6))
sentence2 = ' '.join(Kmers_funct(mySeq2, size=6))
linkcode
#Creating the Bag of Words model:
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform([joined_sentence, sentence1, sentence2]).toarray()
X import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import matplotlib.pyplot as plt
%matplotlib inline
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
human_dna = pd.read_table('../input/dna-sequence-dataset/human.txt')
//LOAD HUMAN DATA
human_dna.head()
human_dna['class'].value_counts().sort_index().plot.bar()
plt.title("Class distribution of Human DNA")
//LOAD CHIMP DATA
chimp_dna = pd.read_table('../input/dna-sequence-dataset/chimpanzee.txt')
chimp_dna.head()
chimp_dna['class'].value_counts().sort_index().plot.bar()
plt.title("Class distribution of Chimpanzee DNA")
//LOAD DOG DATA
dog_dna = pd.read_table('../input/dna-sequence-dataset/dog.txt')
dog_dna.head()
dog_dna['class'].value_counts().sort_index().plot.bar()
plt.title("Class distribution of Dog DNA")
def Kmers_funct(seq, size=6):
    return [seq[x:x+size].lower() for x in range(len(seq) - size + 1)]

#convert our training data sequences into short overlapping k-mers of length 6.
#Lets do that for each species of data we have using our Kmers_funct function.

human_dna['words'] = human_dna.apply(lambda x: Kmers_funct(x['sequence']), axis=1)
human_dna = human_dna.drop('sequence', axis=1)

chimp_dna['words'] = chimp_dna.apply(lambda x: Kmers_funct(x['sequence']), axis=1)
chimp_dna = chimp_dna.drop('sequence', axis=1)

dog_dna['words'] = dog_dna.apply(lambda x: Kmers_funct(x['sequence']), axis=1)
```

```
dog_dna = dog_dna.drop('sequence', axis=1)
human_dna.head()
human_texts = list(human_dna['words'])
for item in range(len(human_texts)):
    human_texts[item] = ' '.join(human_texts[item])
#separate labels
y_human = human_dna.iloc[:, 0].values # y_human for human_dna
chimp_texts = list(chimp_dna['words'])
for item in range(len(chimp_texts)):
    chimp_texts[item] = ' '.join(chimp_texts[item])
#separate labels
y_chim = chimp_dna.iloc[:, 0].values # y_chim for chimp_dna

dog_texts = list(dog_dna['words'])
for item in range(len(dog_texts)):
    dog_texts[item] = ' '.join(dog_texts[item])
#separate labels
y_dog = dog_dna.iloc[:, 0].values  # y_dog for dog_dna
linkcode
y_human
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(ngram_range=(4,4)) #The n-gram size of 4 is previously determined by testing
X = cv.fit_transform(human_texts)
X_chimp = cv.transform(chimp_texts)
X_dog = cv.transform(dog_texts)
linkcode
print(X.shape)
print(X_chimp.shape)
print(X_dog.shape)
# Splitting the human dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y_human,
                                                    test_size = 0.20,
                                                    random_state=42)
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
print("Confusion matrix for predictions on human test DNA sequence\n")
print(pd.crosstab(pd.Series(y_test, name='Actual'), pd.Series(y_pred, name='Predicted')))
def get_metrics(y_test, y_predicted):
    accuracy = accuracy_score(y_test, y_predicted)
    precision = precision_score(y_test, y_predicted, average='weighted')
    recall = recall_score(y_test, y_predicted, average='weighted')
    f1 = f1_score(y_test, y_predicted, average='weighted')
    return accuracy, precision, recall, f1
accuracy, precision, recall, f1 = get_metrics(y_test, y_pred)
print("accuracy = %.3f \nprecision = %.3f \nrecall = %.3f \nf1 = %.3f" % (accuracy, precision, recall, f1))
# Predicting the chimp, dog sequences
y_pred_chimp = classifier.predict(X_chimp)
linkcode
# performance on chimpanzee genes
print("Confusion matrix for predictions on Chimpanzee test DNA sequence\n")
print(pd.crosstab(pd.Series(y_chim, name='Actual'), pd.Series(y_pred_chimp, name='Predicted')))
```

```
accuracy, precision, recall, f1 = get_metrics(y_chim, y_pred_chimp)
print("accuracy = %.3f \nprecision = %.3f \nrecall = %.3f \nf1 = %.3f" % (accuracy, precision, recall, f1))
y_pred_dog = classifier.predict(X_dog)
linkcode
# performance on dog genes
print("Confusion matrix for predictions on Dog test DNA sequence\n")
print(pd.crosstab(pd.Series(y_dog, name='Actual'), pd.Series(y_pred_dog, name='Predicted')))
accuracy, precision, recall, f1 = get_metrics(y_dog, y_pred_dog)
print("accuracy = %.3f \nprecision = %.3f \nrecall = %.3f \nf1 = %.3f" % (accuracy, precision, recall, f1))
```

IV. RESULT

True positive and true negatives are the observations that are correctly predicted and therefore shown in green. We want to minimize false positives and false negatives so they are shown in red color. These terms are a bit confusing. So let’s take each term one by one and understand it fully.

True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted

True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g., if actual class says this passenger did not survive and predicted class tells you the same thing.

False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

False Positives (FP) – When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

False Negatives (FN) – When actual class is yes but predicted class in no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

Once you understand these four parameters then we can calculate Accuracy, Precision, Recall and F1 score.

Accuracy = TP+TN/TP+FP+FN+TN

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

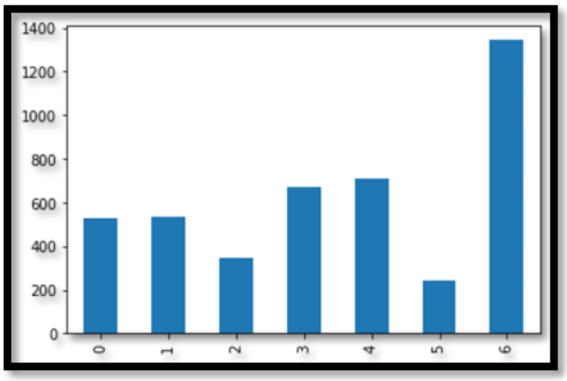
Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it’s above 0.5.

Recall = TP/TP+FN

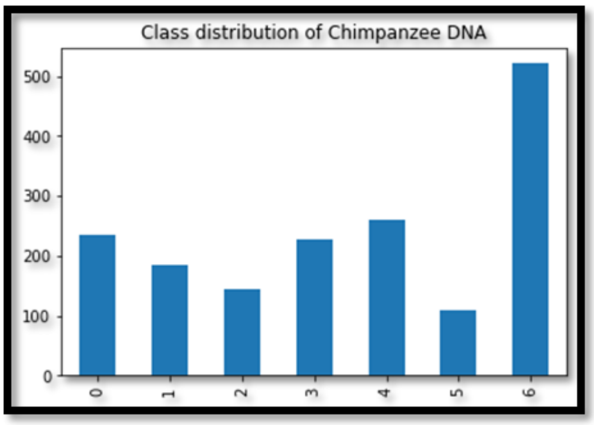
F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall. In our case, F1 score is 0.701.

F1 Score = 2*(Recall * Precision) / (Recall + Precision).

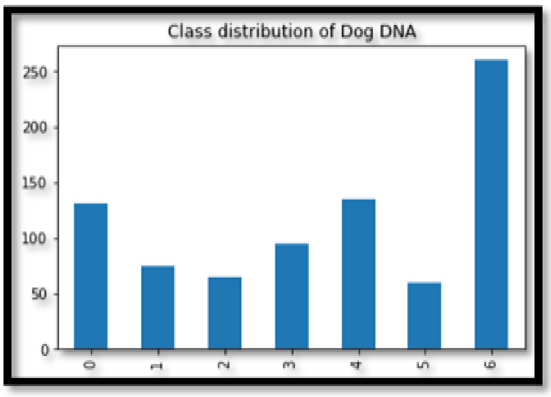
Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model. For our model, we have got 0.803 which means our model is approx. 80% accurate.



CLASS DISTRIBUTION OF HUMAN DNA



CLASS DISTRIBUTION OF CHIMPANZEE DNA



CLASS DISTRIBUTION OF DOG DNA

//CONFUSION MATRIX FOR HUMAN CLASS

```
Output
Confusion matrix

Predicted  0    1     2     3    4    5     6
Actual
0        232    0     0     0    0    0     2
1          0  184     0     0    0    0     1
2          0    0   144     0    0    0     0
3          0    0     0   227    0    0     1
4          2    0     0     0  254    0     5
5          0    0     0     0    0  109     0
6          0    0     0     0    0    0   521
accuracy = 0.993
precision = 0.994
recall = 0.993
f1 = 0.993
```

//CONFUSION MATRIX FOR CHIMP CLASS

```
Output
Confusion matrix
Predicted  0    1    2    3     4     5    6
Actual
0             99    0    0    0     1     0    2
1              0  104    0    0     0     0    2
2              0    0   78    0     0     0    0
3              0    0    0  124     0     0    1
4              1    0    0    0   143     0    5
5              0    0    0    0     0    51    0
6              1    0    0    1     0     0  263
accuracy = 0.984
precision = 0.984
recall = 0.984
f1 = 0.984
```

//CONFUSION MATRIX FOR DOG CLASS

```
Output
Confusion matrix

Predicted  0    1     2     3    4    5     6
Actual
0        127    0     0     0    1    0     4
1          0   63     0     0    1    0    11
2          0    0    49     0    1    0    14
3          1    0     0    81    2    0    11
4          4    0     0     1  126    0     4
5          4    0     0     0    1   53     2
6          0    0     0     1    0    0   260
accuracy = 0.993
precision = 0.994
recall = 0.993
f1 = 0.993
```

V. CONCLUSION

The combination of artificial intelligence technologies such as machine learning and genomics can potentially solve several significant problems we are facing today in medical field.

With powerful machine learning algorithms, genomics researchers will be able to deliver better results faster, at lower cost – making their outcomes available to more people.

SOME OF THE ADVANAGES FROM THE ABOVE RESULTS

Researchers are using machine learning to identify patterns within high volume genetic data sets. These patterns are then translated to computer models which may help predict an individual’s probability of developing certain diseases or help inform the design of potential therapies.

REFERENCES

[1]. S.F. Altschul, , W. Gish, and W. Miller, “A basic local alignment search tool”, J. Mol. Biol. 215, 403–410, 1990.

[2]. R. Begleiter, R. El-Yaniv, and G. Yona, “On prediction using variable order markov models”, Journal of Artificial Intelligence Res., vol. 22, pp. 385–421, 2004.

[3]. N. Cristianini, and J. Shawe-Tahlor, “An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods”, Cambridge University Press, New York, 2002..

[4]. M. Deshpande, and G. Karypis, “Evaluation of techniques for classifying biological sequences”, PAKDD 2002, 417–431, 2002.

[5]. R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, “Biological sequence analysis: Probabilistic Models of Proteins and Nucleic Acids”, Cambridge University Press., 2004.

[6]. S.R Eddy, “Multiple alignment using hidden Markov models”, ISMB 114–120, 1995.

[7]. C. Ferrer-Costa, M. Orozco, X. de la Cruz, “Sequence-based prediction of pathological mutations”.