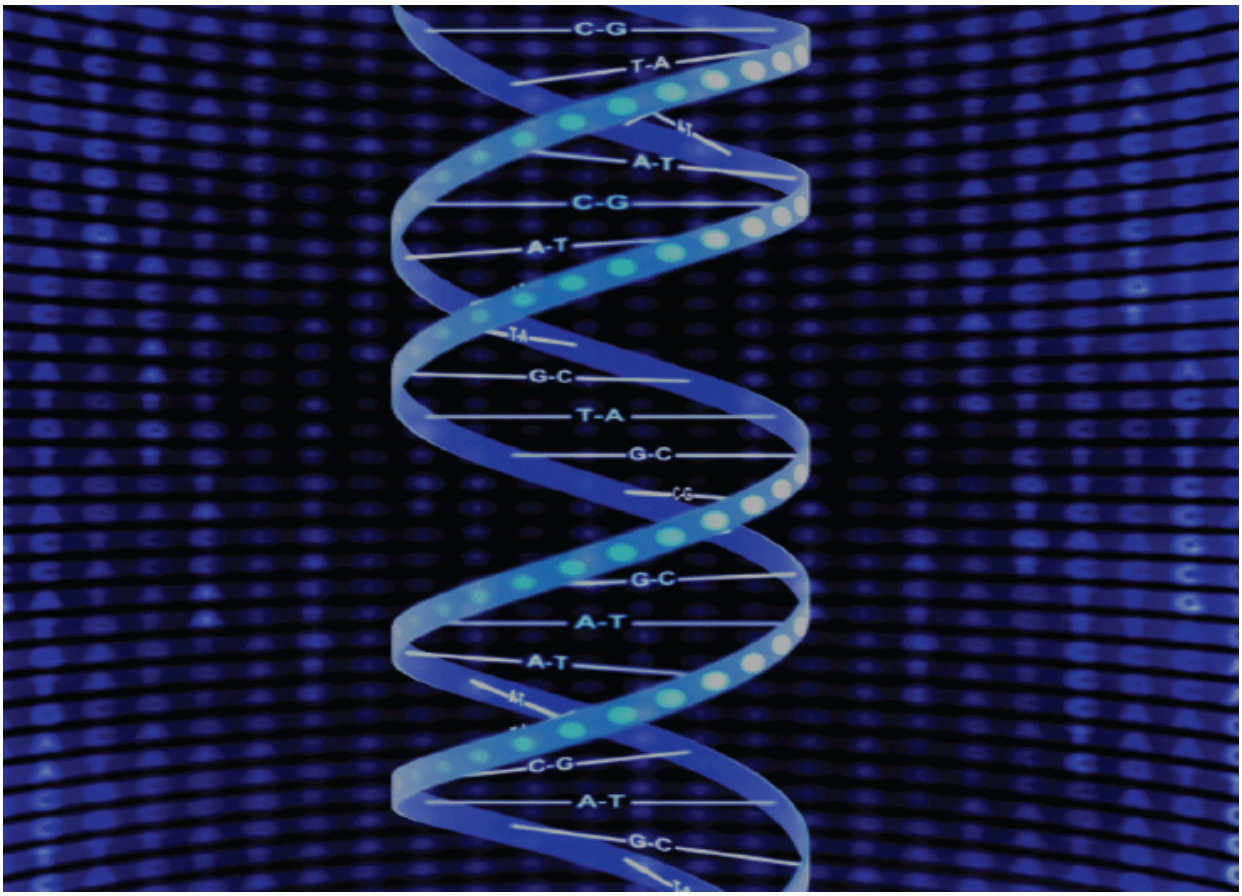
BIOPHYSICS EP-305

DNA SEQUENCING USING MACHINE LEARNING



INNOVATIVE MID-TERM PROJECT

BY **ADITYA SINGH** 2K19/EP/005

AND **ARKAJYOTI CHAKRABORTY** 2K19/EP/022

TABLE OF CONTENTS

DEFINITION

1. Introduction
2. Representation

DNA SEQUENCING

1. History
2. Background

ANALYSIS

1. Data Exploration
2. Exploratory Visualization
3. Algorithms and Techniques
4. Benchmark Model

METHODOLOGY

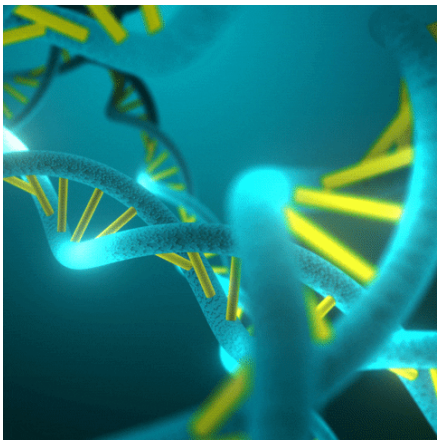
1. Using BioPython
2. Implementation of 3 General Methods
3. Refinement

RESULT

1. Performance Visualization
2. Testing
3. Applications

DEFINITION

A genome is a complete collection of DNA in an organism. All living species possess a genome, but they differ considerably in size. The human genome, for instance, is arranged into 23 chromosomes, which is a little bit like an encyclopedia being organized into 23 volumes. And if you counted all the characters (individual DNA “base pairs”), there would be more than 6 billion in each human genome. So it’s a huge compilation.



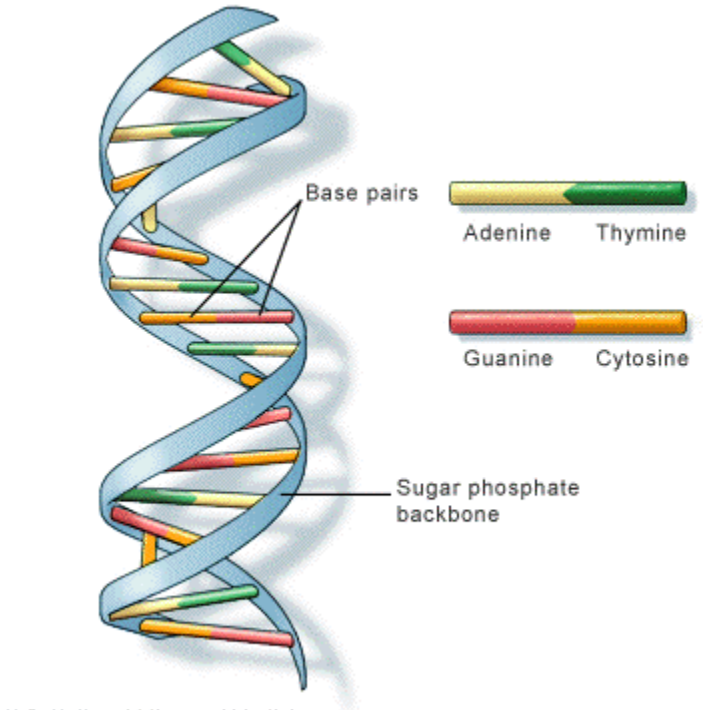
A human genome has about 6 billion characters or letters. If you think the genome(the complete DNA sequence) is like a book, it is a book about 6 billion letters of “A”, “C”, “G” and “T”. Now, everyone has a unique genome. Nevertheless, scientists find most parts of the human genomes are alike to each other.

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 natural language processing. It has become the method of preference for many genomics modeling tasks, including predicting the influence of genetic variation on gene regulatory mechanisms such as DNA receptiveness and splicing.

So we will try to interpret a DNA structure and apply machine learning algorithms to build a prediction model on DNA sequence data.

Representation

The diagram shows a tiny bit of a DNA double helix structure.



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, G and T.

These four chemicals link together via hydrogen bonds in any possible order making a chain, and this gives one thread of the DNA double-helix. And the second thread of the double-helix balances the first. So if you have A on the first thread, you have to have T on the second.

Furthermore, C and G always balance each other. So once you identify one thread of the helix, you can always spell the other.

The order, or sequence, of these bases, determines what biological instructions are contained in a strand of DNA. For example, the sequence ATCGTT might instruct for blue eyes, while ATCGCT might instruct for brown.

DNA SEQUENCING

History

In 1977, Fred Sanger and Alan Coulson published a method to rapidly determine the specific order of the adenine, thymine, cytosine and guanine nucleotides in any DNA sequence. This technology ultimately transformed biology by providing a tool for deciphering complete genes and later entire genomes. Improvements in process parallelization (running hundreds or thousands of samples simultaneously), automation and analysis led to the establishment of factory-like enterprises, called sequencing centers. These facilities spearheaded the effort to sequence the genomes of many organisms, including humans.

Overview

Today, the need for even greater sequencing capability at a more economical price has led to the development of new technologies based on different chemistries and refined for accuracy and speed. These “second generation” approaches reduce the necessary volume of reagents while dramatically increasing the number of simultaneous sequencing reactions in a single experiment. They are capable of producing nearly 150 times more sequence than the first-generation systems, at 1/150th the cost. For example, the cost of sequencing all three billion letters in the human genome has dropped from $15,000,000 to something that is approaching $1,000. The ability to quickly and economically decipher large swaths of DNA has opened doors to research previously deemed out of reach. Many of the discoveries outlined in this guide are in part due to this new technology.

The first so-called “third generation” sequencing system debuted in 2009, producing an entire human sequence. Based on the analysis of a single molecule of DNA, a major technological improvement, it is believed that these systems will become widespread within the next two to three years, further decreasing sequencing costs.

DATA HANDLING

Biopython

Biopython is a set of freely available tools for biological computation written in Python by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the Biopython License, which is extremely liberal and compatible with almost every license in the world.

Here is a brief example of how to work with a DNA sequence in fasta format using Biopython. The sequence object will contain attributes such as id and sequence and the length of the sequence that you can work with directly.

We will use Bio.SeqIO from Biopython for parsing DNA sequence data(fasta). It provides a simple uniform interface to input and output assorted sequence file formats.

```
from Bio import SeqIO
for sequence in SeqIO.parse('../input/dna-sequence-dataset/example_dna.fa', "fasta"):
    print(sequence.id)
    print(sequence.seq)
    print(len(sequence))
```

So it produces the sequence ID, sequence and length of the sequence.

Applying Machine Learning

Now since machine learning or deep learning models require input to be feature matrices or numerical values and currently we still have our data in character or string format. So the next step is to encode these characters into matrices.

There are 3 general approaches to encode sequence data:

1. Ordinal encoding DNA Sequence
2. One-hot encoding DNA Sequence
3. DNA sequence as a “language”, known as k-mer counting

So let us implement each of them and see which gives us the perfect input features.

Ordinal encoding DNA

In this approach, we need to encode each nitrogen bases as an ordinal value. For example “ATGC” becomes [0.25, 0.5, 0.75, 1.0]. Any other base such as “N” can be a 0.

So let us create functions such as for creating a NumPy array object from a sequence string, and a label encoder with the DNA sequence alphabet “a”, “c”, “g” and “t”, but also a character for anything else, “n”.

```
import numpy as np
import re
def string_to_array(seq_string):
    seq_string = seq_string.lower()
    seq_string = re.sub('[^acgt]', 'n', seq_string)
    seq_string = np.array(list(seq_string))
    return seq_string
# create a label encoder with 'acgtn' alphabet
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(np.array(['a','c','g','t','z']))
```

And here is a function to encode a DNA sequence string as an ordinal vector. It returns a NumPy array with A=0.25, C=0.50, G=0.75, T=1.00, n=0.00.

```
def ordinal_encoder(my_array):
    integer_encoded = label_encoder.transform(my_array)
    float_encoded = integer_encoded.astype(float)
    float_encoded[float_encoded == 0] = 0.25 # A
    float_encoded[float_encoded == 1] = 0.50 # C
    float_encoded[float_encoded == 2] = 0.75 # G
    float_encoded[float_encoded == 3] = 1.00 # T
    float_encoded[float_encoded == 4] = 0.00 # anything else, lets say n
    return float_encoded


#Let's try it out a simple short sequence:
seq_test = 'TTCAGCCAGTG'
ordinal_encoder(string_to_array(seq_test))
```

One-hot encoding DNA

Another approach is to use one-hot encoding to represent the DNA sequence. This is widely used in deep learning methods and lends itself well to algorithms like convolutional neural networks. In this example, “ATGC” would become [0,0,0,1], [0,0,1,0], [0,1,0,0], [1,0,0,0]. And these one-hot encoded vectors can either be concatenated or turned into 2-dimensional arrays.

```
from sklearn.preprocessing import OneHotEncoder
def one_hot_encoder(seq_string):
    int_encoded = label_encoder.transform(seq_string)
    onehot_encoder = OneHotEncoder(sparse=False, dtype=int)
    int_encoded = int_encoded.reshape(len(int_encoded), 1)
    onehot_encoded = onehot_encoder.fit_transform(int_encoded)
    onehot_encoded = np.delete(onehot_encoded, -1, 1)
    return onehot_encoded


#So let's try it out with a simple short sequence:
seq_test = 'GAATTCTCGAA'
one_hot_encoder(string_to_array(seq_test))
```

K-mer Counting DNA

A hurdle that still remains is that none of these above methods results in vectors of uniform length, and that is a necessity for feeding data to a classification or regression algorithm. So with the above methods, you have to resort to things like truncating sequences or padding with “n” or “0” to get vectors of uniform length.

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 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 super easy.

```
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)
```

It returns a list of k-mer “words.” You can then join the “words” into a “sentence”, then apply your favorite natural language processing methods on the “sentences” as you normally would.

We have tuned both the word length and the amount of overlap. This allows you to determine how the DNA sequence information and vocabulary size will be important in your application. For example, if you use words of length 6, and there are 4 letters, you have a vocabulary of size 4096 possible words. You can then go on and create a bag-of-words model like you would in NLP.

ANALYSIS

Objective

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.

Gene families are groups of related genes that share a common ancestor. Members of gene families may be paralogs or orthologs. Gene paralogs are genes with similar sequences from within the same species while gene orthologs are genes with similar sequences in different species.

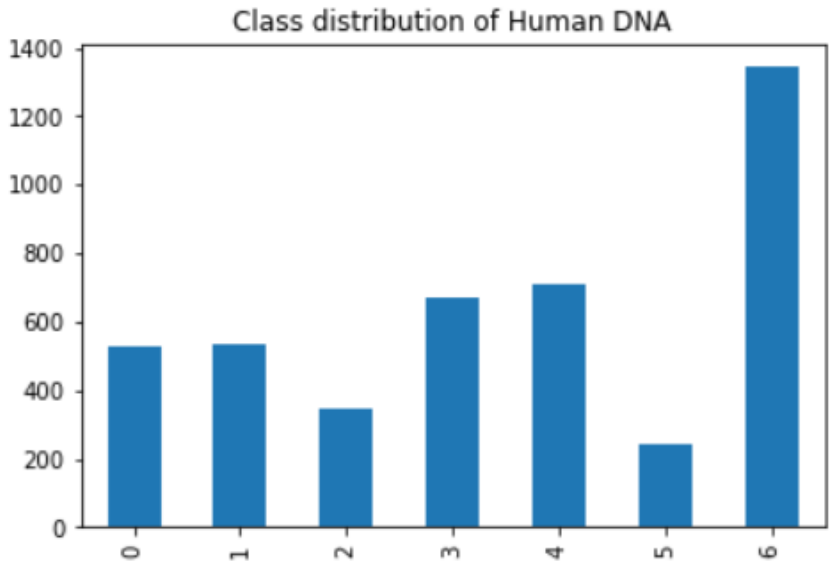
The dataset contains human DNA sequence, Dog DNA sequence, and Chimpanzee DNA sequence.

```
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))
```

Load human DNA data

```
human_dna = pd.read_table('../input/dna-sequence-dataset/human.txt')
human_dna.head()
```

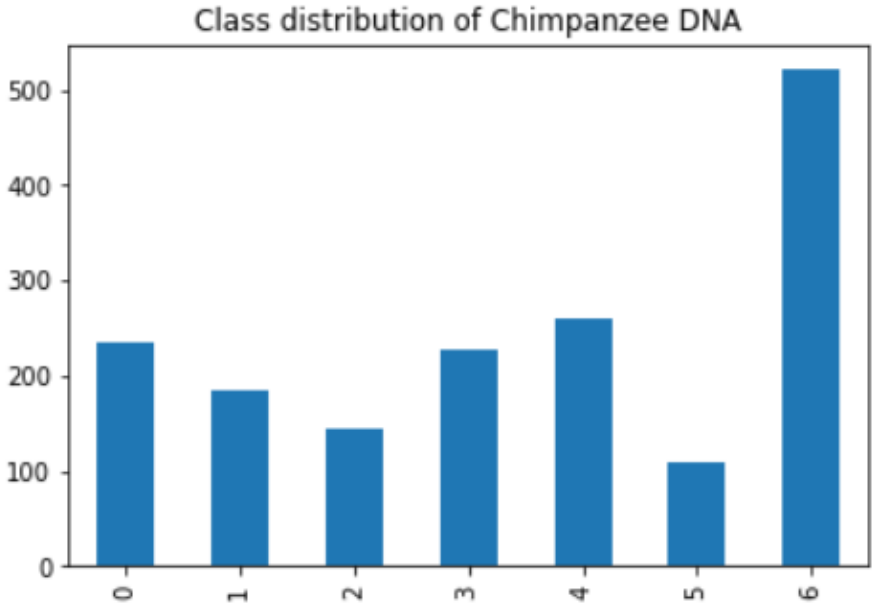
Distribution :



Load Chimpanzee DNA data

```
chimp_dna = pd.read_table('../input/dna-sequence-dataset/chimpanzee.txt')
chimp_dna.head()
```

Distribution :



Here are the definitions for each of the 7 classes and how many there are in the human training data:

Gene family	Number	Class label
G protein coupled receptors	531	0
Tyrosine kinase	534	1
Tyrosine phosphatase	349	2
Synthetase	672	3
Synthase	711	4
Ion channel	240	5
Transcription factor	1343	6

Now we have all our data loaded, the next step is to convert a sequence of characters into k-mer words, default size = 6 (hexamers). The function Kmers_funct() will collect all possible overlapping k-mers of a specified length from any sequence string. Now we have all our data loaded, the next step is to convert a sequence of characters into k-mer words, default size = 6 (hexamers).

```
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)
```

The DNA sequence is changed to lowercase, divided into all possible k-mer words of length 6, and ready for the next step.

Naive Bayes Classifier

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.

So now that we know how to transform our DNA sequences into uniform length numerical vectors in the form of k-mer counts and ngrams, we can now go ahead and build a classification model that can predict the DNA sequence function based only on the sequence itself.

Here I will use the human data to train the model, holding out 20% of the human data to test the model. Then we can challenge the model's generalizability by trying to predict sequence function in other species (the chimpanzee and dog).

Next, train/test split human dataset and build simple multinomial naive Bayes classifier.

You might want to do some parameter tuning and build a model with different ngram sizes, here I'll go ahead with an ngram size of 4 and a model alpha of 0.1.

```
# 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)
```

We will create a multinomial naive Bayes classifier. I previously did some parameter tuning and found the ngram size of 4 (reflected in the Countvectorizer() instance) and a model alpha of 0.1 did the best.

```
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB(alpha=0.1)
classifier.fit(X_train, y_train)
```

RESULTS

Now let's make predictions on the human hold out test set and see how it performs on unseen data

Okay, so let's look at some model performance metrics like the confusion matrix, accuracy, precision, recall and f1 score. We are getting really good results on our unseen data, so it looks like our model did not overfit to the training data. In a real project I would go back and sample many more train test splits since we have a relatively small data set.

```
y_pred = classifier.predict(X_test)
```

```
Confusion matrix for predictions on human test DNA sequence

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
```

Now for the real test. Let's see how our model performs on the DNA sequences from other species. First we'll try the Chimpanzee, which we would expect to be very similar to humans. Then we will try man's (and woman's) best friend, the Dog DNA sequences.

```
Confusion matrix for predictions on Chimpanzee test DNA sequence

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 predictions on Dog test DNA sequence

Predicted    0   1   2   3    4   5    6
Actual
0          127   0   0   0    0   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   0    0   0  260
accuracy = 0.926
precision = 0.934
recall = 0.926
f1 = 0.925
```

The model seems to produce good results on human data. It also does on Chimpanzee which is because the Chimpanzee and humans share the same genetic hierarchy. The performance of the dog is not quite as good which is because the dog is more diverging from humans than the chimpanzee.

APPLICATIONS

Knowledge of the sequence of a DNA segment has many uses. First, it can be used to find genes, segments of DNA that code for a specific protein or phenotype. If a region of DNA has been sequenced, it can be screened for characteristic features of genes.

For example, open reading frames (ORFs)—long sequences that begin with a start codon (three adjacent nucleotides; the sequence of a codon dictates amino acid production) and are uninterrupted by stop codons (except for one at their termination)—suggest a protein-coding region.

Also, human genes are generally adjacent to so-called CpG islands—clusters of cytosine and guanine, two of the nucleotides that make up DNA.

If a gene with a known phenotype (such as a disease gene in humans) is known to be in the chromosomal region sequenced, then unassigned genes in the region will become candidates for that function.

Second, homologous DNA sequences of different organisms can be compared in order to plot evolutionary relationships both within and between species.

Third, a gene sequence can be screened for functional regions. In order to determine the function of a gene, various domains can be identified that are common to proteins of similar function.

For example, certain amino acid sequences within a gene are always found in proteins that span a cell membrane; such amino acid stretches are called transmembrane domains.

If a transmembrane domain is found in a gene of unknown function, it suggests that the encoded protein is located in the cellular membrane.

Other domains characterize DNA-binding proteins. Several public databases of DNA sequences are available for analysis by any interested individual.

END