MINI-PROJECT

AIM: Classifying the deoxyribonucleic acid (DNA**)** using a Support vector Machine and Logistic Regression Algorithms.

SOFTWARE USED: Google Collaboratory

THEORY:

1) Basic knowledge of DNA:

DNA, or deoxyribonucleic acid, is the hereditary material in humans and almost all other organisms. Nearly every cell in a person's body has the same DNA. Most DNA is located in the cell nucleus (where it is called nuclear DNA), but a small amount of DNA can also be found in the mitochondria. Mitochondria are structures within cells that convert the energy from food into a form that cells can use. The information in DNA is stored as a code made up of four chemical bases: adenine (A), guanine(G), cytosine (C), and thymine (T). Human DNA consists of about 3 billion bases, and more than 99 percent of those bases are the same in all people. The order, or sequence, of these bases, determines the information available for building and maintaining an organism, similar to the way in which letters of the alphabet appear in a certain order to form words and sentences.

DNA bases pair up with each other, A with T and C with G, to form units called base pairs. Each base is also attached to a sugar molecule and a phosphate molecule. Together, a base, sugar, and phosphate are called a nucleotide. Nucleotides are arranged in two long strands forming a double helix spiral.

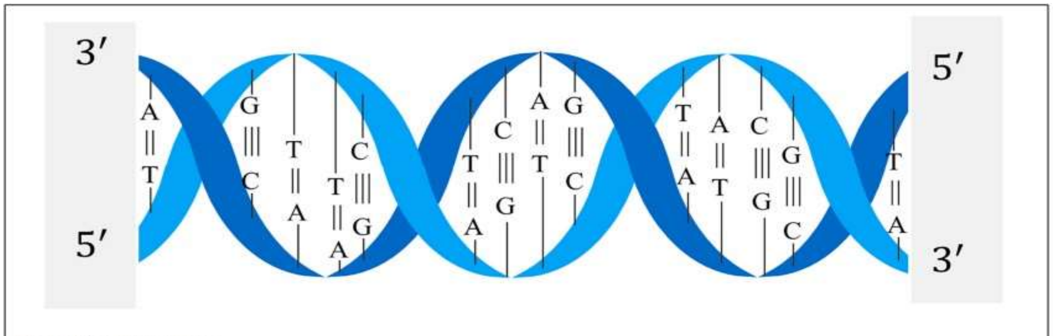


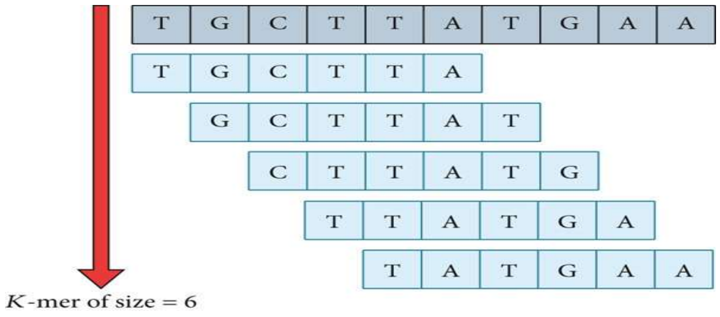
FIGURE 2 | Double helix of DNA.

2) Basic ways of encoding DNA sequence:

1. Sequence coding - The coding region of a gene, also known as the coding sequence (CDS), is the portion of a gene's DNA or RNA that codes for protein. Studying the length, composition, regulation, splicing, structures, and functions of coding regions compared to non-coding regions over different species and time periods can provide significant information regarding gene organization and evolution of prokaryotes and eukaryotes.
2. **One-hot coding:** One-hot encoding is a way to represent categorical data as binary vectors. Machine learning algorithms cannot work with categorical data directly.

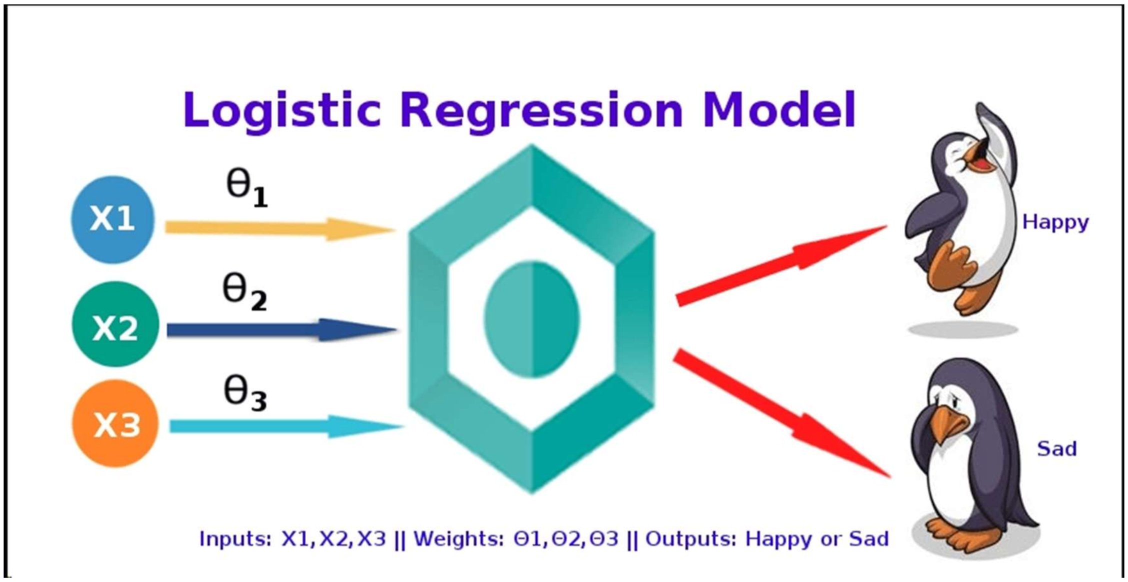
Instead, we use character-level one-hot encoding to represent the categorical data(for us DNA). One hot encoding is a way to represent categorical data as binary vectors. Machine learning algorithms cannot work with categorical data directly. Instead, we use character-level one-hot encoding to represent the categorical data(for us DNA).

3. **K-mer coding**: In recent years, K-mer-based analysis and comparison methods have become standard tools for the analysis of large DNA sequences such as chromosomes, whole genomes, or even metagenomes. The big advantage of K-mer-based methods compared to alignment-based methods such as the well-established National Center for Biotechnology Information (NCBI) Basic Local Alignment Search Tool (BLAST) software family, is the shorter computation times or more precisely, the better scaling of computation times with sequence length. For many purposes, standard K-mer methods deliver reliable results. However, there are also cases where they are still very unsatisfying when compared with those of other methods, for example, during the determination of the phylogenetic distance between two genomes, where the alignment of short DNA motifs such as highly conserved ribosomal RNA genes delivers very reliable results, while the results of K-mer methods are often uncertain [3]. Perhaps the main difference between the results of an alignment-based and K-mer-based method when used on the same data set (e.g., comparison of two genome sequences), is that the results of the alignment method can include the exact position (in bp) and quality of similarity of every part of the sequence within the data set. In contrast, the standard K-mer-method only interprets the sequences as “bags of words, “ neglecting any positional information [4]. The result of a standard K-mer analysis is mostly only comprised of a number representing the similarity between each pair of sequences. To improve the results of K-mer methods, it seems reasonable to add positional resolution to the analysis, while maintaining the advantage of a faster computation time.



3. Machine Learning algorithms:

Logistic Regression - Logistic Regression is a Machine Learning method that is used to solve classification issues. It is a predictive analytic technique that is based on the probability idea. The classification algorithm Logistic Regression is used to predict the likelihood of a categorical dependent variable. The dependent variable in logistic regression is a binary variable with data coded as 1 (yes, True, normal, success, etc.) or 0 (no, False, abnormal, failure, etc.). A Logistic Regression model is similar to a Linear Regression model, except that the Logistic Regression utilizes a more sophisticated cost function, which is known as the “Sigmoid function” or “logistic function” instead of a linear function.


Logistic Regression Model
X1
X2
X3
θ1
θ2
θ3
Happy
Sad
Inputs: X1,X2,X3 || Weights: Θ1,Θ2,Θ3 || Outputs: Happy or Sad

Logistic Regression can be expressed as,

$$log\left(\frac{p(X)}{1-p(X)}\right) = \beta 0 + \ \beta 1X$$

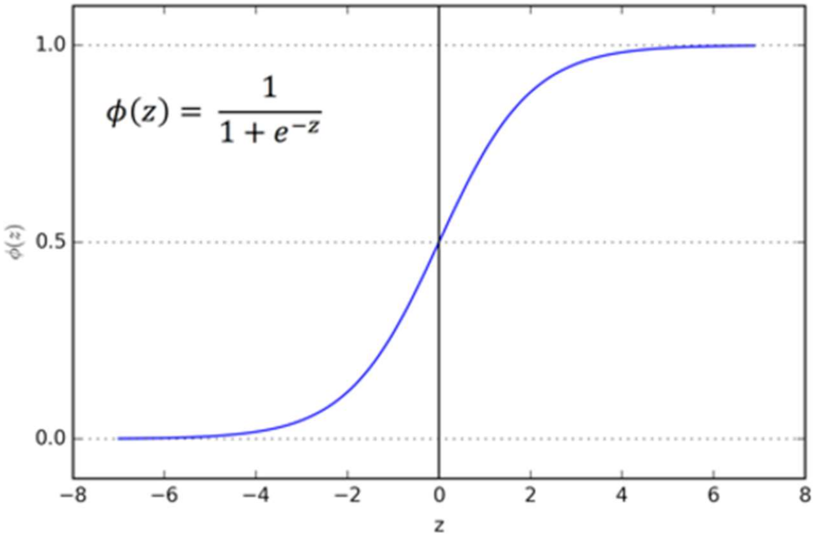
where p(x)/(1-p(x)) is termed odds, and the left-hand side is called the logit or log-odds function. The odds are the ratio of the chances of success to the chances of failure. As a result, in Logistic Regression, a linear combination of inputs is translated to log(odds), with an output of 1.

The following is the inverse of the aforementioned function

$$p(X) = \frac{e^{\beta 0+\beta 1X}}{1+\ e^{\beta 0+\beta 1X}}$$

This is the Sigmoid function, which produces an S-shaped curve. It always returns a probability value between 0 and 1. The Sigmoid function is used to convert expected values to probabilities. The function converts any real number into a number between 0 and 1. We utilize sigmoid to translate predictions to probabilities in machine learning. The mathematical Sigmoid function can be

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



3) Support Vector Machine:

Support vector machines are a set of supervised learning methods used for classification, regression, and outliers detection. All of these are common tasks in machine learning. You can use them to detect cancerous cells based on millions of images or you can use them to predict future driving routes with a well-fitted regression model. There are specific types of SVMs you can use for particular machine learning problems, like support vector regression (SVR) which is an extension of support vector classification (SVC). The main thing to keep in mind here is that these are just math equations tuned to give you the most accurate answer possible as quickly as possible. SVMs are different from other classification algorithms because of the way they choose the decision boundary that maximizes the distance from the nearest data points of all the classes. The decision boundary created by SVMs is called the maximum margin classifier or the maximum margin hyperplane.

Types of SVMs: There are two different types of SVMs, each used for different things:

- Simple SVM: Typically used for linear regression and classification problems.
- Kernel SVM: Has more flexibility for non-linear data because you can add more features to fit a hyperplane instead of a two-dimensional space.

SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data

4) Count Vectorizer:

Machines cannot understand characters and words. So when dealing with text data we need to represent it in numbers to be understood by the machine. A count vectorizer is a method to convert text to numerical data.

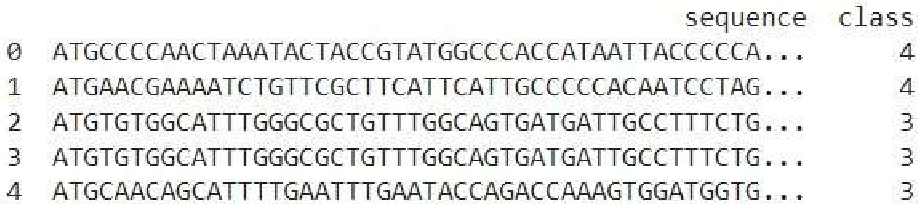
5) Applications of Machine Learning in DNA Sequences:

a) A machine learning approach to optimizing cell-free DNA sequencing panels: with an application to prostate cancer.BMC cancer.

b) A Digital DNA Sequencing Engine for Ransomware Detection Using Machine Learning

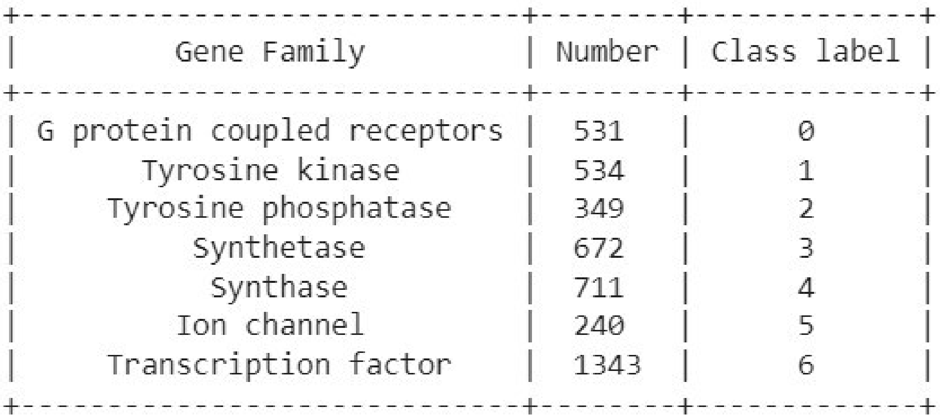
RESULTS:

- Data for human DNA sequence coding regions:



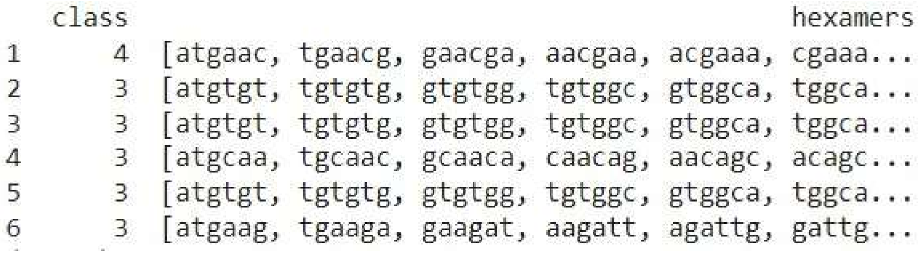
```
                                            sequence  class
0  ATGCCCCAACTAAATACTACCGTATGGCCCACCATAATTACCCCCA...      4
1  ATGAACGAAAATCTGTTCGCTTCATTCATTGCCCCCACAATCCTAG...      4
2  ATGTGTGGCATTTGGGCGCTGTTTGGCAGTGATGATTGCCTTTCTG...      3
3  ATGTGTGGCATTTGGGCGCTGTTTGGCAGTGATGATTGCCTTTCTG...      3
4  ATGCAACAGCATTTTGAATTTGAATACCAGACCAAAGTGGATGGTG...      3
```

- Definitions for each of the 7 classes are in the human training data. These are gene sequence function groups:



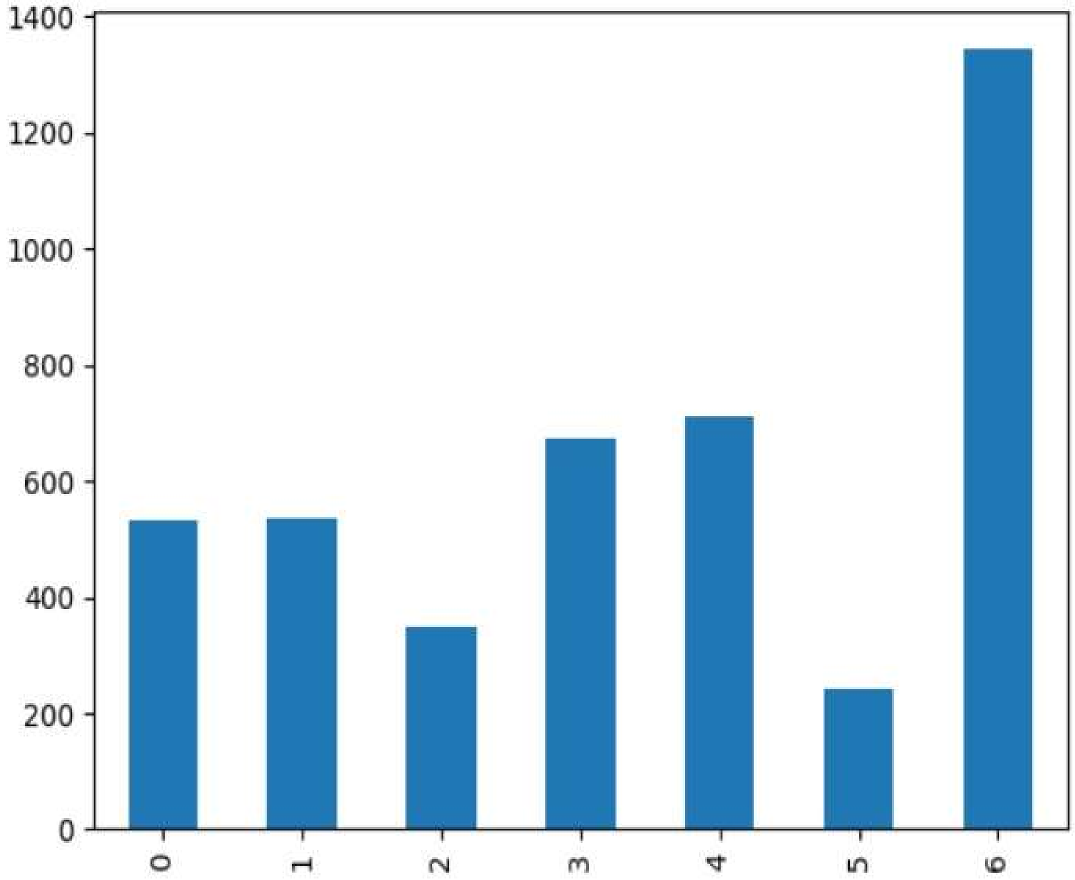
```
+-----------------------------+--------+-------------+
|         Gene Family         | Number | Class label |
+-----------------------------+--------+-------------+
| G protein coupled receptors |  531   |      0      |
|       Tyrosine kinase       |  534   |      1      |
|     Tyrosine phosphatase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

- Coding sequence data is changed to lowercase, split up into all possible k-mer words(hexamers) of length 6 :



```
   class                                          hexamers
1      4  [atgaac, tgaacg, gaacga, aacgaa, acgaaa, cgaaa...
2      3  [atgtgt, tgtgtg, gtgtgg, tgtggc, gtggca, tggca...
3      3  [atgtgt, tgtgtg, gtgtgg, tgtggc, gtggca, tggca...
4      3  [atgcaa, tgcaac, gcaaca, caacag, aacagc, acagc...
5      3  [atgtgt, tgtgtg, gtgtgg, tgtggc, gtggca, tggca...
6      3  [atgaag, tgaaga, gaagat, aagatt, agattg, gattg...
```

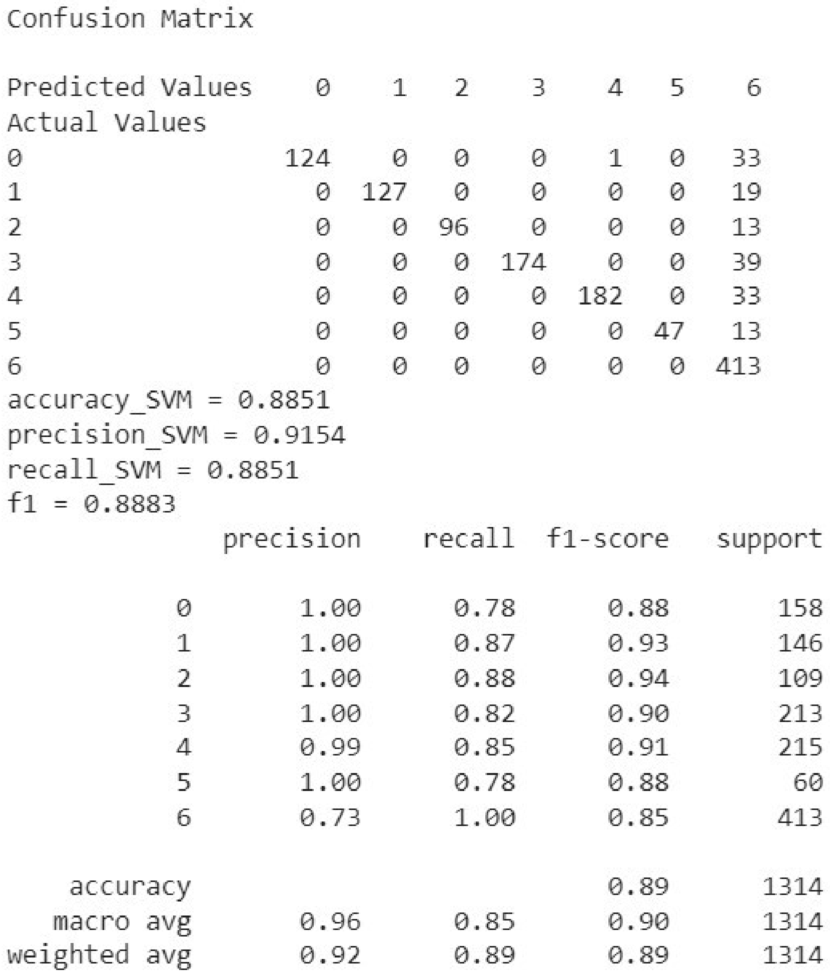
- Class balance dataset:



X-Axis = number of classes
Y-Axis = available data in respective classes

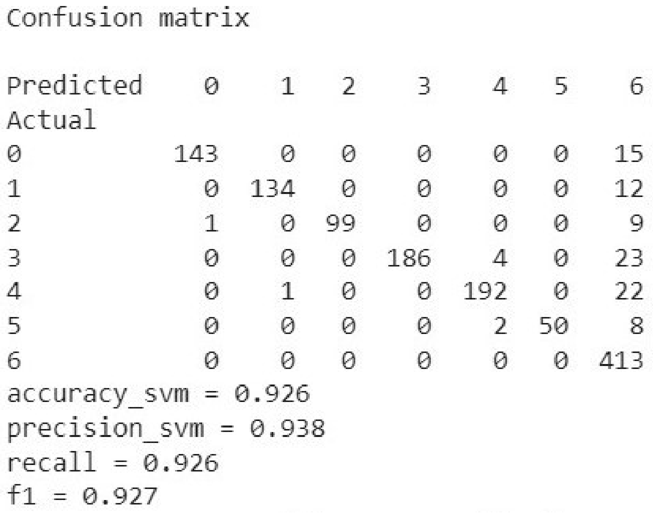
- Confusion matrix for SVM and LOGISTIC REGRESSION model:

SVM Model:

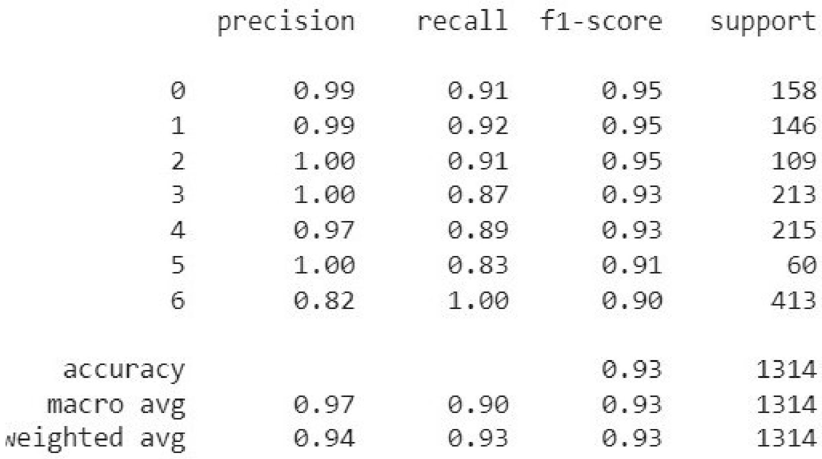


```
Confusion Matrix

Predicted Values    0    1   2    3    4   5    6
Actual Values
0                 124    0   0    0    1   0   33
1                   0  127   0    0    0   0   19
2                   0    0  96    0    0   0   13
3                   0    0   0  174    0   0   39
4                   0    0   0    0  182   0   33
5                   0    0   0    0    0  47   13
6                   0    0   0    0    0   0  413
accuracy_SVM = 0.8851
precision_SVM = 0.9154
recall_SVM = 0.8851
f1 = 0.8883
              precision    recall  f1-score   support

           0       1.00      0.78      0.88       158
           1       1.00      0.87      0.93       146
           2       1.00      0.88      0.94       109
           3       1.00      0.82      0.90       213
           4       0.99      0.85      0.91       215
           5       1.00      0.78      0.88        60
           6       0.73      1.00      0.85       413

    accuracy                           0.89      1314
   macro avg       0.96      0.85      0.90      1314
weighted avg       0.92      0.89      0.89      1314
```

Logistic Regression Model:



```
Confusion matrix

Predicted    0    1   2    3    4   5    6
Actual
0          143    0   0    0    0   0   15
1            0  134   0    0    0   0   12
2            1    0  99    0    0   0    9
3            0    0   0  186    4   0   23
4            0    1   0    0  192   0   22
5            0    0   0    0    2  50    8
6            0    0   0    0    0   0  413
accuracy_svm = 0.926
precision_svm = 0.938
recall = 0.926
f1 = 0.927
```



```
              precision    recall  f1-score   support

           0       0.99      0.91      0.95       158
           1       0.99      0.92      0.95       146
           2       1.00      0.91      0.95       109
           3       1.00      0.87      0.93       213
           4       0.97      0.89      0.93       215
           5       1.00      0.83      0.91        60
           6       0.82      1.00      0.90       413

    accuracy                           0.93      1314
   macro avg       0.97      0.90      0.93      1314
weighted avg       0.94      0.93      0.93      1314
```

DISCUSSION:

- In this project, we presented a support vector machine and logistic regression algorithms for classifying DNA sequences. This method comprises three phases
- k-mer counting: I first take the long biological sequence and break it down into k-mer lengths "ATGCATGCA" becomes: 'ATGCAT', 'TGCATG', 'GCATGC', 'CATGCA'. Hence our example sequence is broken down into 4 hexamer words.
- Creating the Bag of Words model using Count Vectorizer (). This is equivalent to k-mer counting.
- SVM and Logistic regression algorithms are created and the data can be split into training and testing and applied to these two algorithms
- 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 the training data.
- For SVM Model: Accuracy = 0.8851,Recall =0.8851,Precision = 0.9154,F1 score = 0.883.
- For Logistic Regression: Accuracy = 0.926,Recall =0.8851,Precision = 0.938,F1 score = 0.927
- Logistic regression performed well than SVM for DNA classification.

CONCLUSION:

Implemented the Support Vector Machine and Logistic Regression model algorithms to classify the DNA sequences

CODE:

```
import sys
import numpy
import sklearn
import pandas
import numpy as npp
import pandas as pdd
import matplotlib.pyplot as plt
%matplotlib inline


path = '/content/human.txt'
data_human = pdd.read_table(path)
data_human["sequence"][10]
print(data_human.head())
from prettytable import PrettyTable
Table = PrettyTable(["Gene Family", "Number", "Class label"])
Table.add_row(["G protein coupled receptors", "531", "0"])
Table.add_row(["Tyrosine kinase", "534", "1"])
Table.add_row(["Tyrosine phosphatase", "349", "2"])
Table.add_row(["Synthetase", "672", "3"])
Table.add_row(["Synthase", "711", "4"])
Table.add_row(["Ion channel", "240", "5"])
Table.add_row(["Transcription factor", "1343", "6"])
print(Table)

def get_Kmers(sequence, size=6):
   return [sequence[p:p+size].lower() for p in range(len(sequence) -
 size + 1)]

data_human['hexamers'] =data_human.apply(lambda p: get_Kmers(p['sequ
ence']), axis=1)
data_human =data_human.drop('sequence', axis=1)
print(data_human[1:7])
human_list = data_human["hexamers"].values.tolist()
#print(human_list[1])
for i in range(len(human_list)):
  separator = ' '
  human_list[i] = separator.join(human_list[i])
#print(human_list[1])
class_data = data_human.iloc[:,0].values
print(class_data.shape)
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(4,4))
X_human = vectorizer.fit_transform(human_list)
data_human['class'].value_counts().sort_index().plot.bar()
from sklearn.model_selection import train_test_split
X_Train, X_Test, y_Train, y_Test = train_test_split(X_human, class_d
ata, test_size = 0.30, random_state=50)
```

1.Support Vector Machine Model:

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,f1
_score, precision_score, recall_score
cls = SVC(C=100).fit(X_Train, y_Train)
y_Predict = cls.predict(X_Test)
print("Confusion Matrix\n")
print(pdd.crosstab(pdd.Series(y_Test, name='Actual Values'), pdd.Ser
ies(y_Predict, name='Predicted Values')))
def get_values(y_Test, y_Predicted):
    Recall = recall_score(y_Test, y_Predicted, average='weighted')


    Precision = precision_score(y_Test, y_Predicted, average='weight
ed')
    Accuracy = accuracy_score(y_Test, y_Predicted)
    F1_score = f1_score(y_Test, y_Predicted, average='weighted')
    return Accuracy, Precision, Recall, F1_score
Accuracy, Precision, Recall, F1_score = get_values(y_Test, y_Predict
)
print("accuracy_SVM = %.4f \nprecision_SVM = %.4f \nrecall_SVM = %.4
f \nf1 = %.4f" % (Accuracy, Precision, Recall, F1_score))
print(classification_report(y_Test, y_Predict))
```

2. LOGISTIC REGRESSION:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score,f1
_score, precision_score, recall_score
lda = LDA(n_components= 2)

lg = LogisticRegression(random_state= 0)
lg.fit(X_train,y_train)
y_pred = lg.predict(X_test)
print("Confusion matrix\n")
print(pdd.crosstab(pdd.Series(y_test, name='Actual'), pdd.Series(y_p
red, name='Predicted')))
def get_metrics(y_test, y_predicted):
    accuracy = accuracy_score(y_test, y_predicted)
    precision = precision_score(y_test, y_predicted, average='weight
ed')
    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_svm = %.3f \nprecision_svm = %.3f \nrecall = %.3f \n
f1 = %.3f" % (accuracy, precision, recall, f1))
print(classification_report(y_test, y_pred))
```