Big Data Mining in Healthcare
Assignment -2
Group-15

Richa Dwivedi(MT20104)

Shreya Garg(MT20133)

Reshan Faraz(Phd19006)

Submission id:
Shreya Garg
shreya20133@iiitd.ac.in

Submitted To:
Dr. GPS Raghava

NOTE:

1. **We've submitted an A2_MT20133_Phd19006_MT20104.ipynb file to run the code and predicted CSV file.**
2. **The language used: Python.**
3. **We have used the libraries pandas, NumPy, matplotlib, sklearn, etc.**
4. **kaggle competition link: https://www.kaggle.com/t/eef896d3115a4e2281f85ab9bdc9ad97**
5. **Team Name- Group16**

PROBLEM STATEMENT:

Classify Interacting and Non-Interacting RNA Sequences on given data.

File Description:

- RNA_Train..csv:

  Dataset size -(, 2)
  Target variable - 'label'

- text.csv:

  This is a test dataset containing fields in which we have to predict the target variable "label".

- ➢ sample_submission.csv: A sample submission file in the correct format containing fields Sequence and label.

Command level options:

The below code shows the command level options:

```
inFile1=""
inFile2=""
outFile=""


#input separator -i
input_sep1=sys.argv[1]
inFile1 = sys.argv[2]
input_sep2=sys.argv[3]
inFile2 = sys.argv[4]
output_sep=sys.argv[5]
outFile = sys.argv[6]
```

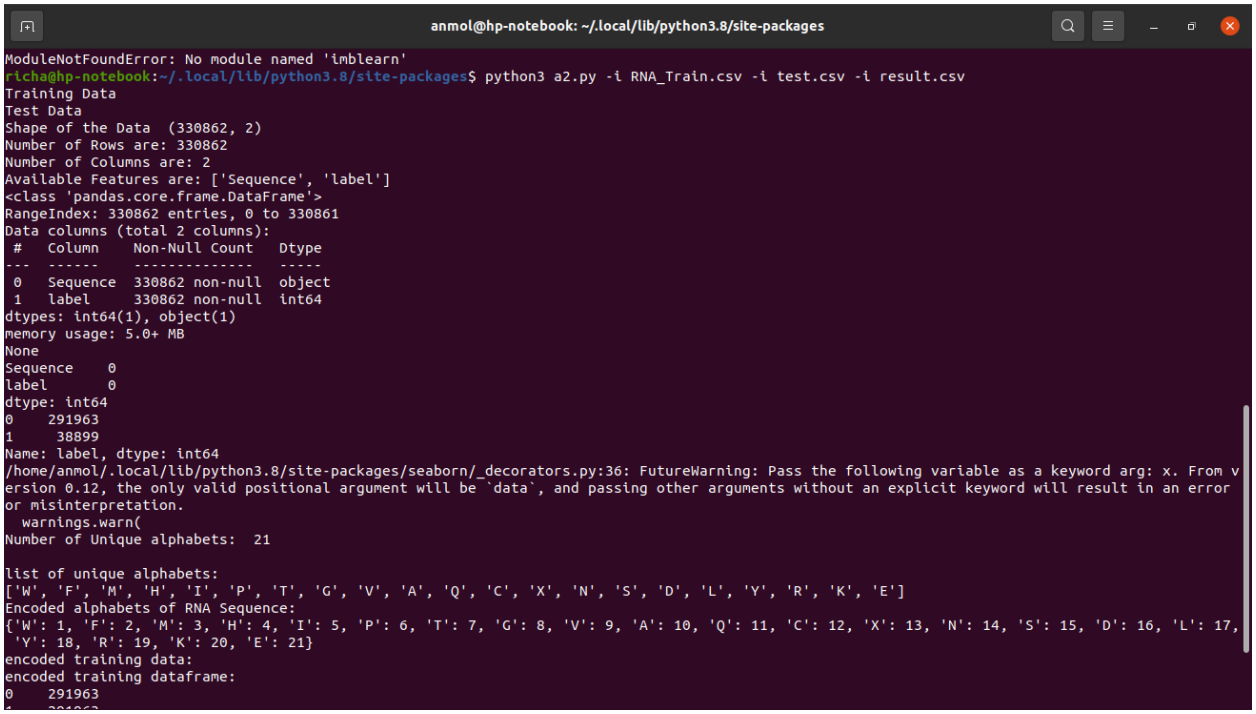
Here,

1. input_sep1 : -i
2. inFile1 : RNA_Train.csv
3. input_sep2 : -i
4. inFile2 : test.csv
5. output_sep : -i
6. outFile : result.csv

The python command used in terminal is shown below:

```
python3 a2.py -i RNA_Train.csv -i test.csv -i result.csv
```

Output:



METHODOLOGY:

1. Libraries used:

Firstly importing all the required libraries, which we are going to use further.
For Ex: pandas, NumPy, seaborn, RandomForestClassifier, etc.

2. Loading Training and testing Data:

- ➢ Firstly extracting all the zip files using zipObj.extractall()
- ➢ Then using pd.read_csv to load, train and test data files.
- ➢ Loading RNA_Train.csv and test.csv files in the train and test data respectively.

```
def loading_dataset():

    with ZipFile('RNA_Train.csv.zip', 'r') as zipObj:
        zipObj.extractall()
    #Reading CSV Files
    train= pd.read_csv('RNA_Train.csv')
    test= pd.read_csv('test.csv')

    return train, test
```

3. Data Overview:

➢ Normal Structure of data.

```
train.head(7)
```

output:

	Sequence	label
0	XXXXXXXXMLQLVRAGA	0
1	XXXXXXXMLQLVRAGAR	0
2	XXXXXXMLQLVRAGART	0
3	XXXXXMLQLVRAGARTW	0
4	XXXXMLQLVRAGARTWF	0
5	XXXMLQLVRAGARTWFR	0
6	XXMLQLVRAGARTWFRP	0

➢ The shape of the data, which represents the number of elements in each dimension.

```
print("Shape of the Data ", train.shape)
```

➢ The number of rows in the dataset.

```
#Number of Rows in the dataset
print("Number of Rows are:",train.shape[0])
```

➢ The number of columns in the dataset.

```
#Number of Columns/features in dataset
print("Number of Columns are:",train.shape[1])
```

➢ List of Available Features in the dataset.

```
#List of Available features in dataset
print("Available Features are:",train.columns.tolist())
```

output:

```
Shape of the Data  (330862, 2)
Number of Rows are: 330862
Number of Columns are: 2
Available Features are: ['Sequence', 'label']
```

➢ Checking types of Value are Available in Features.

```
# Checking types of Value are Available in Features.
print(train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 330862 entries, 0 to 330861
Data columns (total 2 columns):
 #   Column    Non-Null Count   Dtype 
---  ------    --------------   ----- 
 0   Sequence  330862 non-null  object
 1   label     330862 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 5.0+ MB
None
```

➢ Checking if data contains any null/nan values

```
#Checking if data contains any null/nan values
print(train.isnull().sum())
```

output:

data contains no null values.

```
Sequence    0
label       0
dtype: int64
```

4. Data Preprocessing:

i. Data Encoding

The RNA sequence is originally present in String format. (eg. XXXXXXXXSEVSDTNLY)
In order to apply Machine learning techniques, the data should be properly encoded into numeric form. We have performed the following steps:

a. Finding unique list and total number of unique alphabets, among all given sequences in training set

```
list_single_pattern=[]
for seq in train['Sequence']:
 list_single_pattern.append(list(set(seq)))

flat_list = [item for sublist in list_single_pattern for item
in sublist]
newlist=list(set(flat_list))

print("Number of Unique alphabets: ",len(newlist))
print("\nlist of unique alphabets:")
print(newlist)
```

b. Converting the obtained list of alphabets in dictionary and assigning unique number as label to each alphabet

```
#Converting the list of alphabets in dictionary an assigning
unique number as label to each alphabet

unique_dict = {}
for index, val in enumerate(newlist):
   unique_dict[val] = index+1
```

c. Now Encoding sequences in a train and test data into numerical values, in the form of arrays..

```
def encode_data(df):

 encoded_list = []
 for i in df['Sequence'].values:
   encode_val = []
   for value in i:
     encode_val.append(unique_dict.get(value, 0))
   encoded_list.append(np.array(encode_val))
  return encoded_list
```

d. Converting the obtained array into the dataframe.

```
train_df=pd.DataFrame(train_encoded)
train_df["label"]=train["label"]

#Converting test data
test_df=pd.DataFrame(test_encoded)
```

e. obtained data after encoding.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	label
0	11	11	11	11	11	11	11	11	3	4	20	4	6	13	1	16	1	0
1	11	11	11	11	11	11	11	3	4	20	4	6	13	1	16	1	13	0
2	11	11	11	11	11	11	3	4	20	4	6	13	1	16	1	13	2	0
3	11	11	11	11	11	3	4	20	4	6	13	1	16	1	13	2	8	0
4	11	11	11	11	3	4	20	4	6	13	1	16	1	13	2	8	12	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
330857	1	18	17	14	16	2	12	18	4	6	1	3	4	11	11	11	11	0
330858	18	17	14	16	2	12	18	4	6	1	3	4	11	11	11	11	11	0
330859	17	14	16	2	12	18	4	6	1	3	4	11	11	11	11	11	11	0
330860	14	16	2	12	18	4	6	1	3	4	11	11	11	11	11	11	11	0
330861	16	2	12	18	4	6	1	3	4	11	11	11	11	11	11	11	11	0

330862 rows × 18 columns

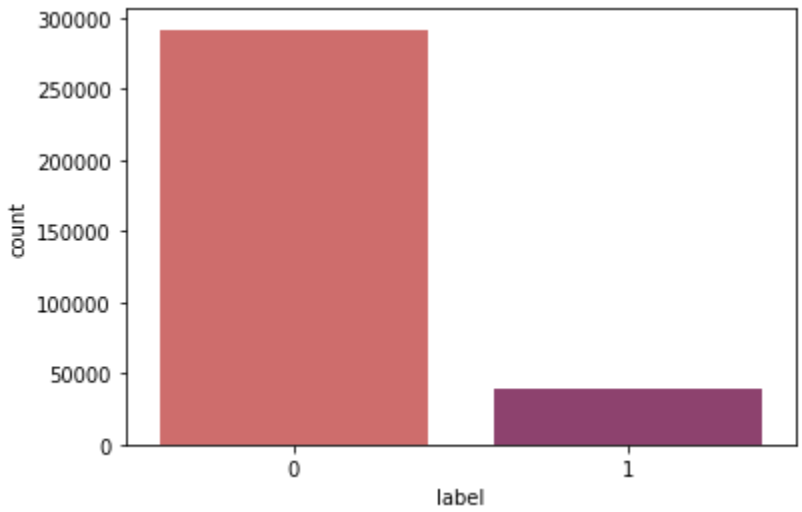
ii. Data Balancing

The unique class labels for identifying interacting or non-interacting RNA are classified as 0 and 1.

```
print(train["label"].value_counts())
# since there are 291963 values of class 0 and 38899 values of class
1.
#so the class is highly unbalanced
```

The class labels are imbalanced as shown in the count plot below:

```
#Visualization of unbalanced class
sns.countplot(train["label"],palette="flare")
```



The above count plot shows that the `291963` data records belong to class 0 whereas `38899` data records belong to class 1.

Hence, the class imbalance has to be addressed using the Sampling technique. We have applied the Upsampling technique where the number of data records for the minority class is increased and balanced with that of the majority class.

```
"""DATA BALANCING"""

def data_balancing(df):
```

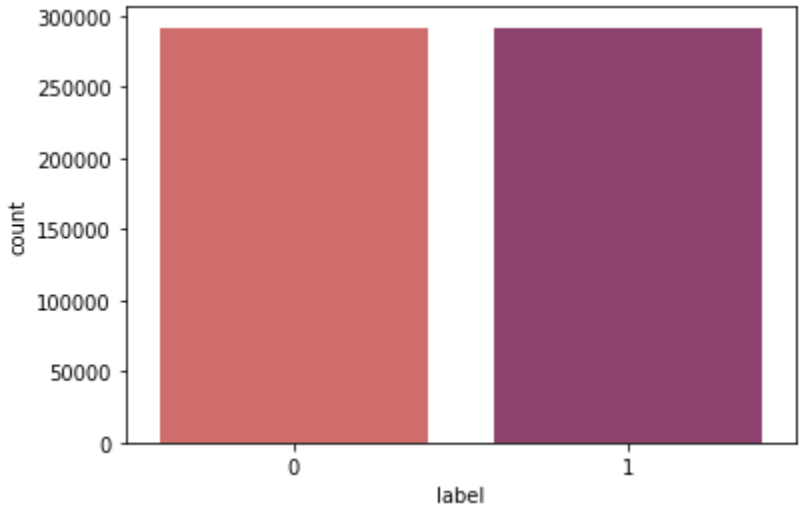
```
    df_majority = df[df.label==0]
    df_minority = df[df.label==1]

    # # Upsample minority class
    traindf = resample(df_minority,
                                        replace=True,     # sample with
replacement
                                        n_samples=df_majority.shape[0],
# to match majority class
                                        random_state=42) # reproducible
results

    # Combine majority class with upsampled minority class
    train1 = pd.concat([df_majority, traindf])

    return train1
```

Hence, after upsampling, we obtain a total of `291963` data records for both class 1 and for class 0.



Obtained data after data balancing contains 583926 rows and 18 columns.

5. Approach used:

steps:

- After Data encoding and balancing, firstly dividing the data into x_train and y_train where x_train all the available features which will be used for training and y_train contains the target variable.

```
def split(df):
   y_train=df["label"]
   x_train=df.drop("label",axis=1)

   return x_train,y_train

x_train,y_train=split(train1)
```

- After splitting the data tried various variations of various models for training.

  Random Forest:

  It is a type of supervised learning technique, used for classification and regression. It contains multiple decision trees which are designed from various parts of the dataset. and improves prediction accuracy by taking the average. and also applied hyperparameter turning using GridSearchCV to find the best parameters.

```
def train_data(x_train,y_train):
   clf = RandomForestClassifier(n_estimators=100)
   clf.fit(x_train,y_train)
joblib.dump(clf,
'/content/drive/MyDrive/bdmh_assignment2/random_forest.pkl')

   return clf

model=train_data(x_train,y_train)
```

  Naive Bayes:

  It is a type of supervised learning technique, which is based on the Bayes theorem.

Gradient Boost Classifier:

It is a type of ensemble learning technique, which improve accuracy of the system by reducing error obtained from the previous result, and this classification method try to improve the prediction accuracy of weak learners

Logistic Regression: It is a type of supervised learning technique, which is used for classification tasks. It works by modelling the probability of occurrence of each class for predicting the class labels.

Others:
Other models such as K-Nearest-Neighbors classification, Decision Tree, SVM were also tried upon on the training set.

6. Results:

S No.	Approach used	MCC score
1	**RandomForest with modified label encoding and upsampling**	**0.84999**
2	**RandomForest with feature selection(12 best features) and upsampling**	**0.84718**
3	**RandomForest with n_estimators=1000**	**0.84437**
4	**RandomForest without upsampling and one-hot encoding**	**0.83041**
5	**RandomForest with upsampling and one-hot encoding**	**0.82905**
6	**NaiveBayes**	**0.61169**
7	**Logistic Regression**	**0.51167**

7. Learning

- Studied the protein interaction between RNA sequences
- Preprocessing large dataset (more than three lakh records)

- Converting RNA Sequence from String format to numeric format to make it fit for classification

- Studied and applied different machine learning & deep learning-based classifiers to preprocessed data

8. Conclusion

Classification of RNA Sequences into interacting and non-interacting forms is important to study protein interactions. Different classification models have been applied on the given data after applying appropriate preprocessing techniques to the raw data. Random forest outperformed among all other classifiers. Random Forest works quite well when applied to a large dataset. Hyperparameter tuning is also performed in order to improve the performance of Random Forest Classifier.

9. References

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/text/one_hot

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html