README

1. Predict whether the Gene has function “Cell Communication” Using micro expression data and functional data Using various Classification techniques.

As the problem statement suggests, the objective of this project was to accurately predict whether the Gene has function “Cell Communication” using micro expression data and functional data available in the “Ecoli.csv” dataset using various Classification techniques. But before we can apply the classification on the data we have to check the if the data is consistent, complete and usable in its current state. If the data has issues, we will have to solve these issues first by using various pre-processing techniques like data imputation, normalization and feature engineering. The Classification algorithms used here are Decision Tree, K-Nearest Neighbors, Random Forest and Naïve Bayes. To improve the performance of these models we will use K-fold Cross Validation with Grid Search.

1.1 Understanding the data

This dataset contains 107 columns of which 103 are numerical features and three columns are nominal features (column 104 to 106). The final column(107th) is the target column which indicates the label of the dataset representing the “Cell communication”. Here, “1” denotes the presence of “Cell communication” and “0” indicates its absence. This dataset has 1500 rows.

```
In [2]: df=pd.read_csv("Ecoli.csv")
        df.head()
```

Out[2]:

ol 4)	Num (Col 5)	Num (Col 6)	Num (Col 7)	Num (Col 8)	Num (Col 9)	Num (Col 10)	...	Num (Col 98)	Num (Col 99)	Num (Col 100)	Num (Col 101)	Num (Col 102)	Num (Col 103)	Nom (Col 104)	Nom (Col 105)	Nom (Col 106)	Target (Col 107)
09	-0.196312	0.024514	1.169839	0.398493	0.569329	0.247633	...	1.933925	-0.094288	0.235789	-0.026187	-0.483784	NaN	0.0	0.0	0.0	0
71	-0.080745	-0.059902	-0.763669	0.038235	-0.192427	-0.692135	...	-0.352500	-0.503082	-1.856162	-0.013567	0.057459	0.087260	0.0	0.0	0.0	0
10	-0.070239	-1.336964	-0.782618	0.082714	0.031681	-1.073164	...	0.445367	0.354260	-0.708162	-0.027058	0.025262	-0.186810	NaN	0.0	0.0	0
iN	0.083342	0.451410	1.116012	-0.217285	-0.317147	NaN	...	0.698107	-0.637167	0.229040	NaN	NaN	-0.230290	0.0	0.0	0.0	0
15	-0.037934	1.923995	-0.601967	-0.548091	-0.071106	0.106609	...	0.499785	-0.134634	NaN	0.022780	NaN	-0.145992	0.0	0.0	0.0	0

1.2 Data Pre-processing

1.2.1 Data Imputation

For data imputation, we used imputation by class-specific value. First we divide the dataset into two datasets by class, i.e, 1 and 0. Then we divide each of these datasets by numerical and ordinal values. So, in total we have four datasets now. Then, for numerical data frames, we use the mean value of all the values present in the column to replace the Nan values. For the ordinal data frames, we replace the Nan values with the most common value for each column. After the process, we don’t have any Nan values and we didn’t lose any data points.

```
In [76]: df.isnull().sum()

Out[76]: Num (Col 1)          0
         Num (Col 2)          0
         Num (Col 3)          0
         Num (Col 4)          0
         Num (Col 5)          0
                             ..
         Nom (Col 104)        0
         Nom (Col 105)        0
         Nom (Col 106)        0
         Target (Col 107)     0
         Anomaly              0
         Length: 108, dtype: int64
```

1.2.2 Anomaly Detection

An anomaly can be defined as something that “deviates” from what is standard or considered as normal. There are five different methods of anomaly detection (Density based technique, Model based technique, Distance based technique, Cluster based technique, Isolation Forest). In our project, we used the “Isolation Forest technique” to find anomalies. In our dataset, we discovered 15 rows with anomalies in them now how they affect our dataset requires further investigation which is out of scope for this project.

```
In [46]: anomalies

Out[46]:
```

ol 0)	...	Num (Col 99)	Num (Col 100)	Num (Col 101)	Num (Col 102)	Num (Col 103)	Nom (Col 104)	Nom (Col 105)	Nom (Col 106)	Target (Col 107)	Anomaly
20	...	-0.631709	0.737136	-0.036406	-0.276895	-31.170580	0.0	0.0	0.0	0	-1
46	...	-0.193928	13.178435	-0.069623	-49.780064	-1.023417	1.0	0.0	0.0	0	-1
30	...	-0.549697	0.131904	-0.020310	25.456065	0.294456	0.0	0.0	0.0	0	-1
18	...	-0.179270	-0.670052	0.000406	0.062211	0.243624	0.0	1.0	0.0	0	-1
27	...	-0.075939	2.314141	0.104817	-0.151158	-0.037944	0.0	1.0	0.0	0	-1
36	...	-0.749486	1.031036	0.040393	-14.136281	-0.456450	0.0	0.0	0.0	0	-1
00	...	-0.329900	-1.458680	-0.053208	-0.206577	-0.332388	0.0	0.0	0.0	0	-1
54	...	0.040441	-0.857261	-10.065948	-0.522653	-0.301466	0.0	0.0	0.0	0	-1
26	...	-0.279887	0.361383	-0.008461	-0.173665	0.545613	0.0	0.0	0.0	0	-1
12	...	-0.063637	-1.125838	0.003897	-0.112781	106.374733	0.0	0.0	0.0	0	-1
36	...	46.621796	-1.566760	-0.073652	-0.114983	1.023593	0.0	0.0	0.0	0	-1
31	...	-0.221155	1.119332	0.002033	0.758175	-0.399771	0.0	0.0	0.0	0	-1
10	...	-0.230226	-0.814392	-0.025775	-0.183565	1.090731	0.0	0.0	1.0	0	-1
49	...	0.406939	-1.545627	0.034516	-0.587922	0.126054	0.0	0.0	1.0	1	-1
53	...	-0.362418	-0.020482	0.068090	-0.501174	-0.165852	0.0	0.0	1.0	1	-1

1.2.3 Normalization

In my project, I used Z-score normalization because Z-score normalization is good at handling outliers (or anomalies) and we have already established that our dataset has 15 anomalies so I decided to use this method for the dataset. In python, Z-score normalization can be used by the **StandardScaler()** function present under **sklearn.preprocessing.** The dataset before normalization looks like this:

```
In [39]: print(X)

[[-0.56477669 -0.01540391  0.11842607 ...  0.          0.
   1.        ]
 [-0.26857507  0.91455836 -0.23050704 ...  0.          0.
   1.        ]
 [-0.76088313  0.77616284 -0.19313252 ...  0.          0.
   1.        ]
 ...
 [-0.66405031 -0.57067081 -0.10605694 ...  0.          1.
   1.        ]
 [ 0.26766973  1.95570925  0.13446865 ...  0.          1.
   1.        ]
 [-0.49123297 -1.21383026  0.13892418 ...  0.          1.
   1.        ]]
```

After Normalization, the dataset looks like:

```
In [41]: from sklearn.preprocessing import StandardScaler
         X=StandardScaler().fit(X).transform(X.astype(float))
```

```
In [42]: print(X)

[[-0.19495828  0.03377212  0.06410731 ... -0.25561868 -0.38427388
   0.10050378]
 [-0.09161853  0.17852298 -0.19392243 ... -0.25561868 -0.38427388
   0.10050378]
 [-0.26337651  0.15698138 -0.16628464 ... -0.25561868 -0.38427388
   0.10050378]
 ...
 [-0.22959317 -0.05265651 -0.10189383 ... -0.25561868  2.60231065
   0.10050378]
 [ 0.09546822  0.34058061  0.07597051 ... -0.25561868  2.60231065
   0.10050378]
 [-0.16930012 -0.15276582  0.07926529 ... -0.25561868  2.60231065
   0.10050378]]
```

1.3 Classification Methods

All Classification algorithms are grouped under Supervised Learning. Supervised Learning means we provide the algorithm with labels for the data so that can learn from already available data and them use this on new data. For this project, we used **Decision Tree, K-Nearest Neighbors, Naïve Bayes** and **Random Forest.** For

Hyperparameter tuning, I used K- Fold Cross Validation with GridSearch. In python GridSearchCV function is present under sklearn.model_selection. In GridSearchCV, a grid of different hyperparameter values is formed and for each combination of these values, a k-fold Cross Validation is performed and the combination with the best score (in my project, I used it on F1 score) is returned. We can use these values to then train the dataset.

1.3.1 Decision Tree

In this classification method, a tree is formed using the features present in the training set. The tree that forms can be explained by two entities, nodes and leaves. The leaves are the final outcomes or decisions that we reach because of the nodes. We use the ID3 algorithm for this project. The best parameters obtained after tuning the model with grid search are criterion as “gini” and max_depth as 2. For the test set created earlier, it gives a f1 score of 0.776.

1.3.2 K-Nearest Neighbours

The main idea of KNN lies in identifying the label of “k” nearest point of the test point and decide the label of the test point by using majority rule. KNN is sometimes referred to as “lazy learning” or “instance based learning” because the training data is just memorized without building any model in training phase. The more similar two objects are, the less distance is between them and this distance is calculated using “Minkowski Distance”. For this project, I am using its most common variant known as Euclidean distance. For this method, we tuned for the best K value and obtained k=7. F1 score obtained for the test set is 0.861 with accuracy score at 0.967.

1.3.3 Random Forest

Random Forest is a type of Ensemble Learning. Ensemble learning is usually considered as an easy and powerful method to achieve to better generalization. In this algorithm, we construct various different decision trees instead of just one and combine all of them together. For this model after tuning we got the best hyperparameters as entropy=”gini” and max_depth=”10”. F1 score obtained for test dataset is 0.878 and accuracy score of 0.973

1.3.4 Gaussian Naïve Bayes

This is probabilistic classification model. It performs classification but estimating posterior distribution, i.e, by estimating $p(y|x)$ where x id the instance and y is the label. For a dataset with two labels y1 and y2, the label for the data point is decided by: $p(y1|x)>p(y2|x)$ than the assigned label is y1 otherwise it is y2. In naïve bayes method, we assure attribute conditional independence which means “Given the class information, all features are conditionally independent from each other”. This model just couldn’t perform well no matter how much tuning was done the best F1 score obtained here is 0.21.

1.4 Evaluation

1.4.1 F1 Score

This is the main evaluation metric used in this project. F1 score was used for tuning as well as validating the model. F1 score can be defined as the harmonic mean between precision and recall. It is between 0 and 1 and higher the score means better the performance will be. For this dataset, F1 scores are as follows:

Model	**F1 Score**
Decision Tree	0.776
KNN	0.861
Random Forest	0.878
Gaussian Naïve Bayes	0.21

Please note that every time we run the code, it returns a slightly different value.

1.4.2 Accuracy Score

This is the second evaluation method used in this project. Accuracy score just calculates the ratio of correct predictions by all labels for this project, Accuracy scores are as follows:

Model	**Accuracy score**
Decision Tree	0.94
KNN	0.967
Random Forest	0.973
Gaussian Naïve Bayes	0.187

Please note that every time we run the code, it returns a slightly different value.

1.5 Conclusion

In conclusion, the best model for this dataset is Random Forest followed by KNN. However, there is no statistically significant difference between the two models. For a completely new dataset either can perform better we cannot really say which one.

2 Dependencies

2.1 Hardware Used

Hardware used for this project:

CPU	Ryzen 7 5800H
GPU	NVIDIA RTX 3060
RAM	16 GB
ROM	1 TB

Please note that during the training process, it took a lot of time to run the gridsearchCV part of the code as it is very GPU heavy. These parts of the code have been changed to "Raw nb convert" so that they don't run while you are running them but they are left in the notebook for your reference. When you run the code, it only runs the tuned models so it won't take a lot of time.

2.2 Software Used

These are the software requirements for this project:

OS	Windows 10
Programming Language	Python 3.8
IDE	Jupyter Notebook (Anaconda)

The following python libraries will also need to be installed:

Numpy

Pandas

Matplotlib

Seaborn

Skit-Learn

Random

These are all the software and libraries that are required to run this code.

3 Authors

Ninaad Akella

[46860051]

Email: n.akella@uqconnect.edu.au

4 License

5 Help

For any advice for common problems or issues contact me through my Email: n.akella@uqconnect.edu.au.

6 Acknowledgements

www.stackoverflow.com

www.github.com

www.geeksforgeeks.com