CS5011 A4 Report

matriculation number: 160021429

1. Overview

The main aim of this practical is to implement the Help-Desk system by using the neural network. The system should use the neural network for the ticket routing process, and help the user to match with the suitable response team that could handle the ticket.

To implement this system, I used python3 and scikit-learn.

1 - 1. Instructions

Assuming that you are using the lab machine, you would need to set up and use the virtualenv to install the required modules such as scikit-learn.

To set up the virtualenv: virtualenv -p /usr/bin/python3.7 venv

To execute the virtualenv: . venv/bin/activate

Now, you need to go to the A4src/ directory and execute “pip install -r requirements.txt” to install all python modules that are specified in the requirements.txt file.

To run the program: python3 A4main.py <Bas | Int | Adv>

Basic agent : python3 A4main.py Bas
Intermediate agent : python3 A4main.py Int
Advanced agent : python3 A4main.py Adv

For the cross validations:

1) K-Fold cross validation for part 1 : python3 part1.py
2) GridSearchCV for the part 1 : python3 part1a.py
3) K-Fold cross validation for part 3 : python3 part3.py

1 - 2. Check table

Section	File name
Basic Agent	part1.py
Intermediate Agent	part2.py
Advanced Agent 1 (Other ML algorithm)	part3.py
Advanced Agent 2 (Retraining with new type of tickets)	part2.py

2. Design and Implementation

2 - 1. Part 1

The aim of the part 1 was to design, build and train a neural network. The first step was to properly encode the training data. Each column header in the csv file (training data file) represents a question, and the subsequent values within that column are either “Yes” or “No”, depending on how the aforementioned question was answered. The final column contains the name of the response team that could help the user.

To successfully train the neural network, the data file must first be encoded into a usable format. Since the question data only has value “Yes” and “No”, I just decided to encode “Yes” to 1 and “No” to 0.

Apparently, we cannot use the same way to encode the last column, since there are more than 2 response teams. However, the scikit-learn provides an ample solution. Within the preprocessing package of the scikit-learn, there is a class called OneHotEncoder. By passing this a dataframe containing the “Response Team” column, it successfully encodes all the names of the response teams with unique values.

As we encoded the data successfully, now we could start building the neural network. The scikit-learn has a class called “MLPClassifier”, which is a classifier with

multiple layers of perceptrons. Since we are only dealing with inputs and outputs of 0's and 1's, a sigmoid function is a valid activation function. With a learning rate starting at 0.5, and a momentum of 0.3, we should avoid any local minimums. The only decision left was the size of the hidden layers.

To decide the size of the hidden layers, initially, I just ran score tests with different values of "hidden_layer_sizes" parameter.

1 hidden layer : accuracy score = 0.384
2 hidden layers : accuracy score = 0.888
3 hidden layers : accuracy score = 0.948
4 hidden layers : accuracy score = 0.992
5 hidden layers : accuracy score = 0.999
10 hidden layers : accuracy score = 1.000

As you could see above, the value of accuracy score is consistently 100% if the number of hidden layers is greater than 5. Therefore, it would be possible to say that we could have the overfitting issue with more than 5 hidden layers.

To make sure that the model is not overfitted, I wrote some codes for cross validation. I will explain more about this in the evaluation section.

2 - 2. Part 2

Part 2 allows the user to answer questions, and the system will accept these answers and attempt the guess the response team that could help the user. To implement this, we were asked to use the neural network that we implemented in the previous section.

At the start, the program will read in the csv file and joblib file. Here, I made an assumption that the joblib file is created by running the basic agent before running the intermediate agent. A new encoder will also made, the same as in part 1 (unfortunately, you cannot use joblib to save encoders). This builds the same encoding map, and will later be used to decode the neural network prediction.

Once all this is done, the system will start asking the questions. These are simply the column header, but with a "?" appended to it. For example, "Request?" to ask question for the Request column. Originally, I had a switch statement, where each column header would output an actual question, like "Is your issue related to the IdCards?", however, I found that when adding new features to the dataset, I would

not be able to create full questions for these new features. With this in mind, I decided to go for a more generalised method, allowing all features within the file to be the output in the question form.

The system will only accept an answer in the form of either ‘Y’ or ‘y’ for “Yes”, and ‘N’ or ‘n’ for “No”.

After the system asked half of questions, it will attempt to make an early guess with the answers it has already gathered. Basically, this is done by substituting all future answers with an answer of “No”.
If the instance occurs that the neural network finds no match to the combination of the answers, it will print “I have no guess..”. If either the system failed to find the matching response team or the user does not happy with the choice of the response team, then the program will ask some additional questions to retrain the neural network.

So, if the system should retrain the neural network, it will ask the name of the response team that the user think they should handle the ticket. The answer of this question will be stored. The system will then ask the user if there is any new feature that the system should know that would help differentiate this new response team with the other teams. If the user says yes, then it will ask the user to input the name of the new feature. This is also stored.

The original training file is then read into a dataframe. If a new feature has been given, a new column is built containing all “No”’s, as it can be assumed that for all current response teams in the database, this feature has never been key for their selection. This new column is then appended to the original dataframe just before the “Response Team” column. If a new feature has not been given, this step can be skipped.

Each new response team and feature the system learns is saved in a file “additions.csv”. This file contains the new addition, whether or not it was a new response team or new feature, and what file it was added to.

Once all this is complete, the system will overwrite the tickets.csv with the new dataframe, and overwrite the joblib file with the new neural network. Then, the system will prepare to process the next ticket, and start asking the questions to the user to handle the next ticket routing.

When implementing the retraining method, I was considering to use “partial_fit”. According to the scikit-learn doc, the MLPClassifier has a member function call “partial_fit”, which works almost same with the fit, however, it updates the model with

a single iteration over the given data. Apparently, we could save time for retraining if the partial_fit works properly. However, since it only iterates once, there is a possibility that the model could have some under fitting issue. Apparently, under fitting is as bad as overfitting issue, thus, I decided to just using the fit rather than partial_fit.

2 - 3. Part 3

Basically, I did 2 extensions - 1) Use, compare and evaluate different training algorithms, and 2) Modify the system with a new view for the IT admin to include additional response teams and tags on the fly.

For the second extension, I accidentally did it while implementing the intermediate agent. I misunderstood the specification, and implemented more than we actually need to do. Basically, we only needed to make the intermediate agent to retrain the neural network with existing response teams. However, I overthought it, and made my intermediate agent to retrain with new response team. Since I already mentioned this in the part 2 section, I will not explain more about it in this section.

For the first extension, I used the RandomForestClassifier class of the scikit-learn. Basically, the RandomForestClassifier uses the “ensemble” method. The ensemble is a method that merges the multiple models to generate some better model. So, the RandomForestClassifier is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset, and uses averaging to improve the predictive accuracy and control over-fitting. This means that the RandomForestClassifier will enhance the accuracy of the model by merging a number of models.

Basically, the “train_classifier_RandomForestClassifier” method in the part3.py file is almost same with the “train_classifier” method in the part1.py file. The only difference is that the train_classifier_RandomForestClassifier method uses the random forest classifier, where the train_classifier uses the MLPClassifier.

The main difference between RandomForestClassifier and MLPClassifier is that the RandomForestClassifier uses the ensemble method to merge a number of weak models to generate some better model. I will mention more about comparison between MLPClassifier and RandomForestClassifier in the evaluation section.

When the program finishes training with the RandomForestClassifier, it will generate a bar chart that plots the feature importance values. Since the RandomForest uses a

number of decision trees, it could calculate the importance of features. So, by using the member attribute “feature_importances_” of the RandomForestClassifier, we can get the values of the feature importances (the higher, the more important the feature). By using this, we can evaluate which feature affects most to the output.

3. Evaluation

To evaluate the performance of neural network, I made 2 test functions: 1) cross_validate, 2) findBestParams_loopN.

The cross_validate function is in the part1.py file, which uses the K-Fold cross validation by using the cross_val_score function of the scikit-learn. The cross_val_score function is a member function of the sklearn.model_selection package, which literally helps the programmer to select the model by analysing the accuracy score by doing the cross validation testing.

To run the cross_validate function, you need to execute the part1.py file by using “python3 part1.py”. By using this, the program will train the neural network with the dataset, and perform the cross validation testing with the generated model to check if the model is overfitted.

As you could see below, my neural network gets some good cross validation scores, which means that it is good enough.

```
Iteration 6250, loss = 0.10732870
Iteration 6251, loss = 0.10731951
Iteration 6252, loss = 0.10731032
Iteration 6253, loss = 0.10730113
Iteration 6254, loss = 0.10729194
Training loss did not improve more than tol=0.000100 for 5000 consecutive epochs. Stopping.
cross_val_score : [0.96428571 0.92771084 0.96385542]
```

The findBestParams_loopN is a function that is in the part1a.py file. The aim of this file is to find the best parameters by using the GridSearchCV. The GridSearchCV does the exhaustive search over specified parameter values for an estimator. The main aim of the GridSearchCV is to do the cross validation to find the best estimator with combinations of the given parameters. So, it is possible to say that the a GridSearchCV is used for finding the best combinations of the hyper-parameters for the given estimator.

Basically, what I did in the findBestParams_loopN function is just run the loop n times, and inside the loop the program executes the GridSearchCV to find the best

parameters. Every single iteration, the program will store the combinations of the best parameters in the list. And after the loop terminates, the findBestParams_loopN function will print out all elements in the list, which are the combinations of the parameters.

```
Print out best parameters
{'activation': 'logistic', 'hidden_layer_sizes': 7, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 9, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 5, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 9, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 8, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
{'activation': 'logistic', 'hidden_layer_sizes': 7, 'learning_rate_init': 0.5, 'max_iter': 1000, 'momentum': 0.3, 'n_iter_no_change':
 5000, 'solver': 'sgd', 'verbose': True}
```

As you could see above, the number of hidden layers is always greater than or equal to 5. This is the main reason that I choose to set the size of hidden layers as 5. Because, when I saw this output, what I thought was that it might be possible to get the overfitting issue by setting the size of the hidden layers with some positive integer that is greater than 5.

By doing the GridSearchCV, I decided to set the number of hidden layers as 5, which is good enough. And by doing the K-Fold cross validation, I was able to check that the neural network that I implemented works properly.

I also did the K-Fold cross validation testing with the RandomForestClassifier. If you type “python3 part3.py”, then you would be able to run the advanced agent with the cross validation testing.

```
perform the cross validation
Start Running the Cross Validation
cross_val_score : [1.         0.96385542 0.97590361]
```

As you could see above, the random forest classifier gets higher score than the MLP classifier. Therefore, it is possible to say that the RandomForestClassifier actually works much better than the MLPClassifier. Perhaps, this is because that the random forest classifier uses the ensemble method to generate better model without having the overfitting issue.

4. Testing

4 - 1. Part 1

Expected: The file is read, the dataframe is succesfully encoded, and a model is successfully trained and tested. This model is then saved in the “output.joblib”.

Actual: The model is successfully trained, scored and saved, suggesting all data was successfully encoded correctly also. Model saved in the “output.joblib”.

```
Iteration 7332, loss = 0.01151504
Iteration 7333, loss = 0.01151469
Iteration 7334, loss = 0.01151199
Iteration 7335, loss = 0.01150974
Iteration 7336, loss = 0.01150965
Iteration 7337, loss = 0.01150877
Iteration 7338, loss = 0.01150417
Iteration 7339, loss = 0.01150389
Iteration 7340, loss = 0.01150426
Iteration 7341, loss = 0.01150077
Iteration 7342, loss = 0.01149818
Training loss did not improve more than tol=0.000100 for 5000 consecutive epochs
. Stopping.

Training score: 1.000000

Saving model as output.joblib
Model saved.
```

4 - 2. Part 2

1) Handling “no guess” case

Expected: When encountering the case where no prediction can be made, the system will print out a plain message, rather than crashing.

Actual: For the both “early prediction” and “normal prediction”, the system can print out a suitable message if the system is not able to make a prediction, rather than crashing.

```
Incident?
Please answer Y for yes, N for no: n
WebServices?
Please answer Y for yes, N for no: y
Login?
Please answer Y for yes, N for no: y
Wireless?
Please answer Y for yes, N for no: n
I think the response team that could help you is Credentials
Am I correct?
        [y/n] >>n
Oh dear. Let's continue...
Printing?
Please answer Y for yes, N for no: y
IdCards?
Please answer Y for yes, N for no: y
Staff?
Please answer Y for yes, N for no: y
Students?
Please answer Y for yes, N for no: n

I have no guess..

Please tell me the name of the Reponse team that could help you:
```

2) Testing known data

Expected: When inputting data already trained on, the system should guess correctly.

Actual: When inputting the first answer set in the “tickets.csv”, it gives the correct answer.

```
Executing Intermediate Agent!
Use newTickets.csv
Request?
Please answer Y for yes, N for no: n
Incident?
Please answer Y for yes, N for no: y
WebServices?
Please answer Y for yes, N for no: y
Login?
Please answer Y for yes, N for no: y
Wireless?
Please answer Y for yes, N for no: y
I think the response team that could help you is Emergencies
Am I correct?
        [y/n] >>y
Cool! I will route you to the Emergencies team right now.
```

3) Testing with random entry data

Expected: The system should be able to make both early prediction and normal prediction.

Actual: The system could make both early prediction and normal prediction. Also, the system could keep answer question if the user answered with invalid input.

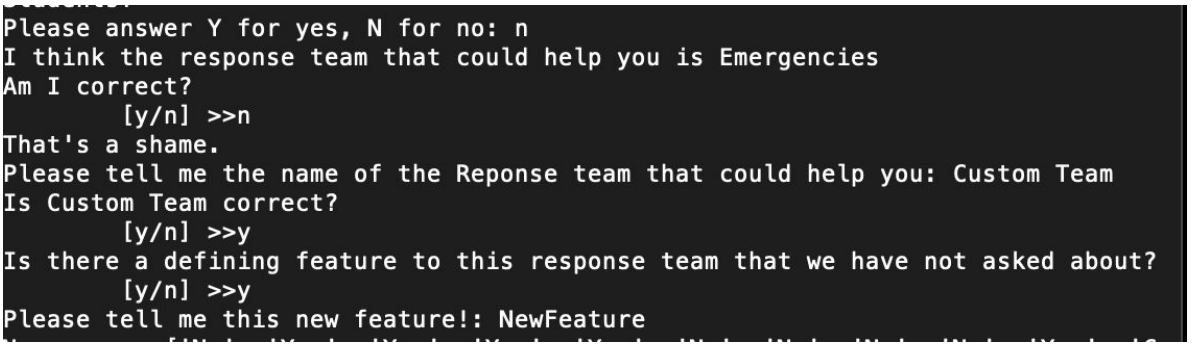
```
Wireless?
Please answer Y for yes, N for no: n
I think the response team that could help you is Networking
Am I correct?
        [y/n] >>n
Oh dear. Let's continue...
Printing?
Please answer Y for yes, N for no: t
Please answer Y for yes, N for no: y
IdCards?
Please answer Y for yes, N for no: y
Staff?
Please answer Y for yes, N for no: n
Students?
Please answer Y for yes, N for no: y
I think the response team that could help you is Equipment
Am I correct?
        [y/n] >>y
Cool! I will route you to the Equipment team right now.
```

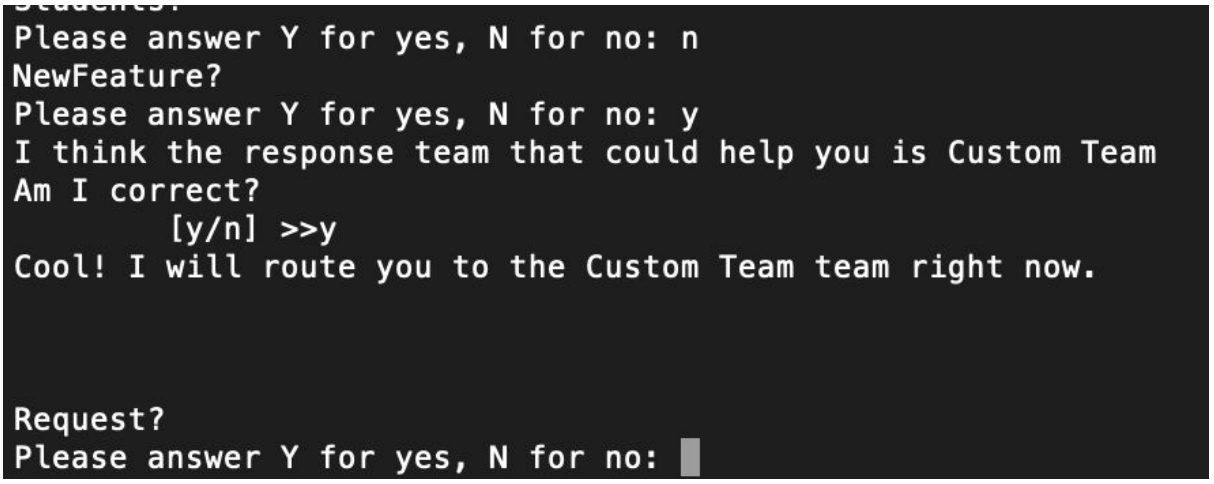
4) Testing addition of new response team

Expected: When told the answer is not correct, the system should asks for new response team and feature. When these are input, the system should retrain the neural network, and add that information to the additions.csv file. When program is ran again, the system should give a correct answer for the user this time.

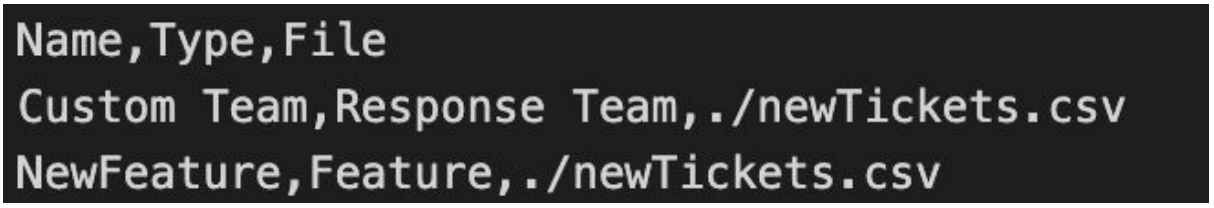
Result: All tests pass. When the user add new feature, then the system will ask question about the new feature on next time. Also, when the system retrains the

neural network, the system could make a prediction with the correct response team, as you could see below.





Also, as you could see below, the system could add new rows to the additions.csv.



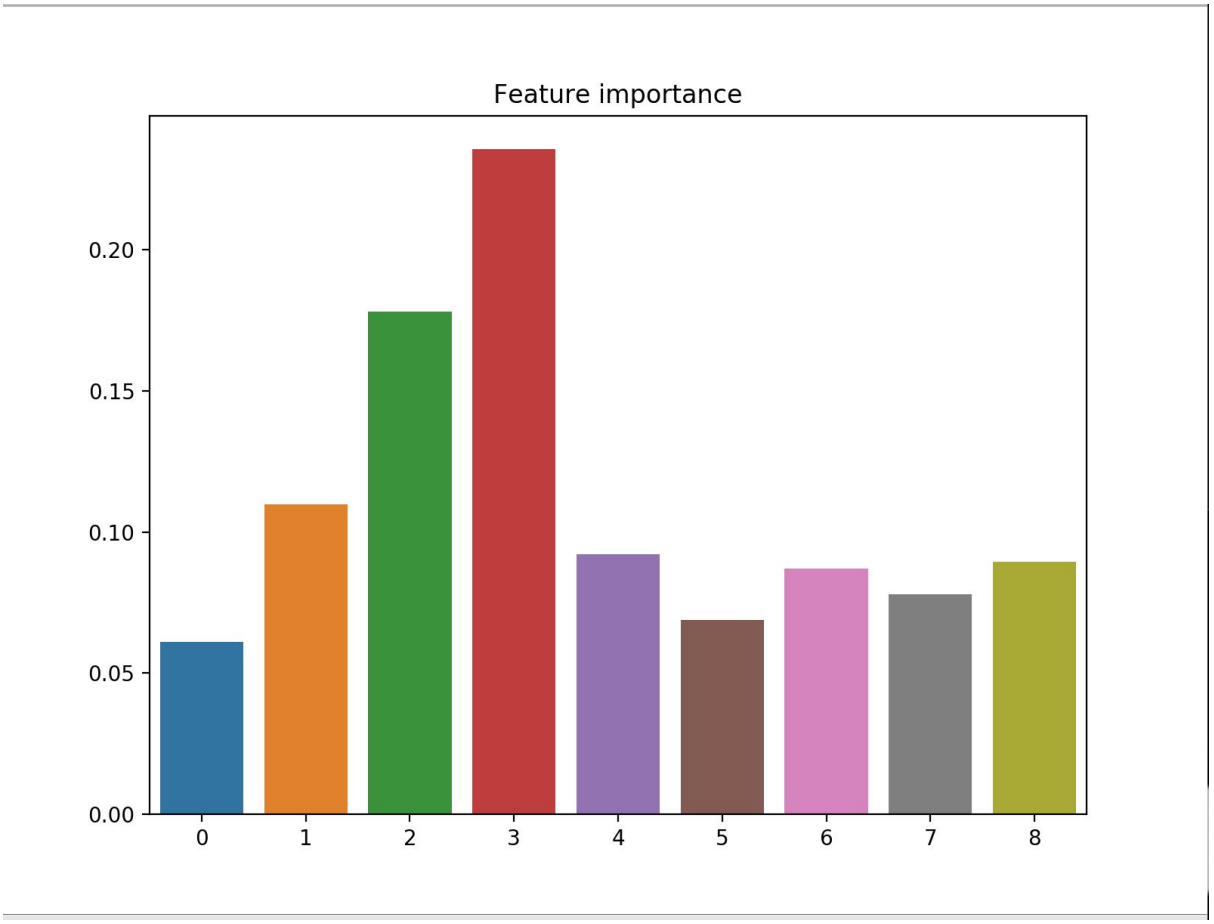
4 - 3. Part 3

1) Training with RandomForestClassifier, and plot the feature importances

Expected: The file is read, the dataframe is succesfully encoded, and a model is successfully trained and tested with the RandomForestClassifier. This model is then saved in the “output_part3.joblib”. Also, the program should generate the bar chart which plots the feature importances.

Actual: The model is successfully trained, scored and saved, suggesting all data was successfully encoded correctly also. Model saved in the “output_part3.joblib”. Also, the program generates the bar chart which plots the feature importances.

```
Creating classifier...
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators='warn',
                       n_jobs=-1, oob_score=False, random_state=3, verbose=0,
                       warm_start=True)

Reading file.

Encoding data...
Data encoded.

Training...

Training score: 1.000000

Saving model as output.joblib
```



5. Conclusion

While doing this practical, I was able to learn how the neural network works by using the MLPClassifier. Also, while doing the extensions, I looked up many other machine learning algorithms such as RandomForest, DecisionTree, XGBoosting, etc. Clearly, it was really helpful for me to learn the basic concepts of the machine learning.