Report on the Investigation

The process starts with the importing of all the necessary libraries such as pandas, seaborn, matplot etc, which allows the user to perform several operations on the data. Once all the necessary libraries are imported the dataset is loaded, this process is called data loading operation. There are two datasets (CE802_P2_Data.csv' and 'CE802_P2_Test.csv) loaded into the machine and named as train and test datasets, these datasets are in the CSV (Comma Separated Values) format and are loaded using the pandas library. The data in these loaded datasets contains some unnecessary or irrelevant values called null values. For identifying these null values, the "isnull().sum()" command is used, this command uses two methods - the isnull() identifies each null value and sum() displays the total number of identified null values present in the attributes of the dataset. In the train dataset, there are 750 null values found in the "F15" column of the dataset and in the test dataset there are no null values found in any attribute. These identified null values are deleted or eliminated using the drop method and the data is prepared for performing the encoding.

The label encoder class is used for performing the conversion from the categorical string values to the categorical integer values and for this conversion the transform method of the label encoder class is used and this transformation of data helps in the effective implementation of the selected models. The "class" attribute of the train dataset contains the string values and using the label encoder class these string values are transformed into the integer format.

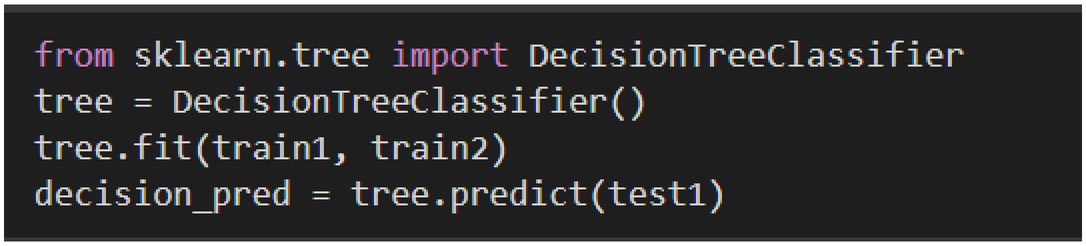
After this using the train dataset (first dataset), the features and target values are selected. All the attributes except the "class" of the train data are selected as the feature values and the "class" attribute is selected as the only target value. The shape of the data which is selected as the feature value is having 1500 rows and 14 columns. Using these selected features and target values, the data is splitted into the two sets of training (train1, train2) and testing data (test1, test2). For this splitting of the dataset, the train_test_split method of the sklearn.model_selection module is used and the data is splitted into 25 % testing data and 75 % testing data.

Implementing the models

Classification models

There are three classification models implemented which are Decision Tree classifier, Random Forest Classifier and Gradient Boosting Classifier.

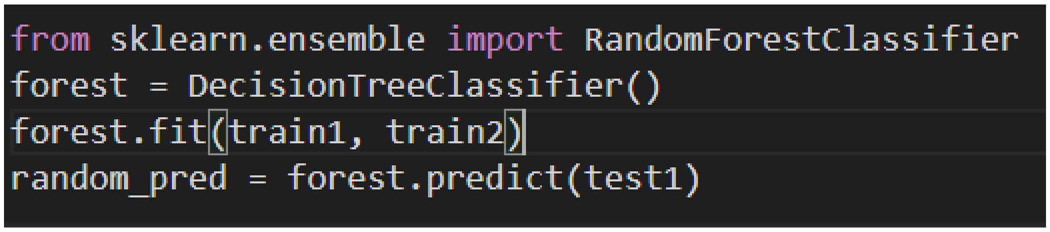
Decision Tree classifier



```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(train1, train2)
decision_pred = tree.predict(test1)
```

This is a classifier model which is used for solving the classification problems, the working of this model is based on the decision tree. The decision tree contains nodes (starting and terminal) and branches. In this model, the predictions are made using a single decision tree. For this the decision tree model is imported from the sklearn.tree library and this imported model is fitted over the two sets of training data (train1 and train2) and using these sets of data the training of the model is performed and this trained model generates predictions using the one set of testing data (test 1).

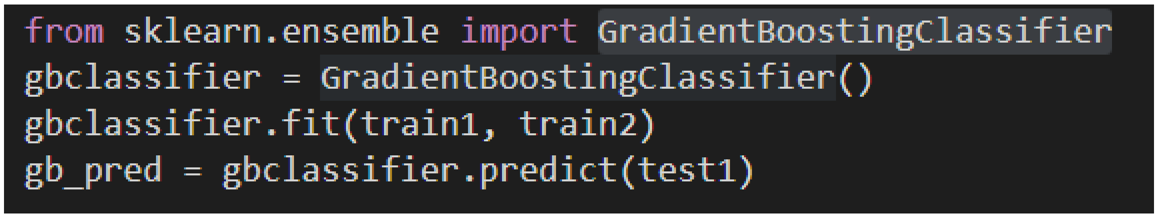
Random Forest Classifier



```
from sklearn.ensemble import RandomForestClassifier
forest = DecisionTreeClassifier()
forest.fit(train1, train2)
random_pred = forest.predict(test1)
```

Random forest classifier is a classification model which is imported from the sklearn.ensemble module of the sklearn library, using the training and testing data the imported model is trained and tested. For this, the training of the model is performed by fitting the model over these two sets of training data (created using the feature and target values) and this trained classifier generates predictions using the one set of the testing data (xtest).

Gradient Boosting Classifier



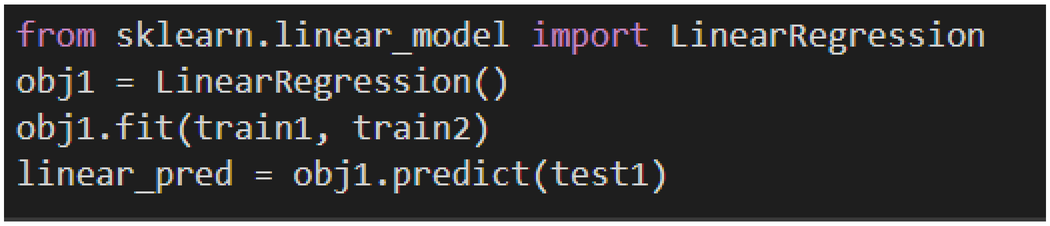
```
from sklearn.ensemble import GradientBoostingClassifier
gbclassifier = GradientBoostingClassifier()
gbclassifier.fit(train1, train2)
gb_pred = gbclassifier.predict(test1)
```

This classifier is a group of ML (machine learning) algorithms which selects different learning models and combines them together in order to create a predictive model (strong model). Using this classifier the loss function value is minimized and for this the function pointing towards the negative gradient is iteratively selected. This ML algorithm is used for performing both the classification and regression tasks. This model is imported from the sklearn.ensemble module and this model is trained using the train1 and train2 sets of training dataset and this trained model generates predictions using the one set of testing data (test1).

Regression models

Three regression models are implemented which are Linear Regression, Decision Tree Regressor and Lasso model.

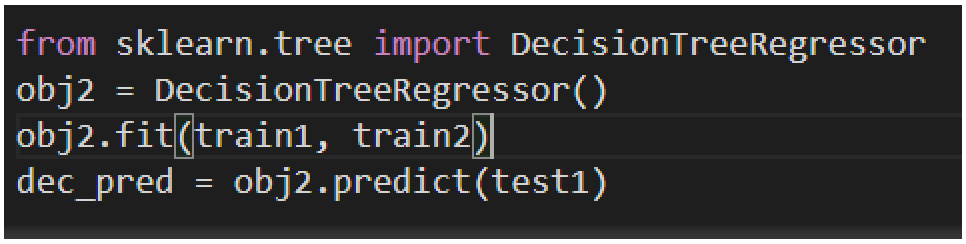
Linear Regression



```
from sklearn.linear_model import LinearRegression
obj1 = LinearRegression()
obj1.fit(train1, train2)
linear_pred = obj1.predict(test1)
```

Linear regression model is used for performing the regression tasks and this model is imported from the sklearn.linear_model library of python and the model is trained by fitting it over the training sets of data (tain1, train2) and for making predictions the one set of testing data (test1) is used by the model . This model uses the one set of the independent variables for generating target prediction values. The relation between the variables of the data and forecasting is found using the linear regression model.

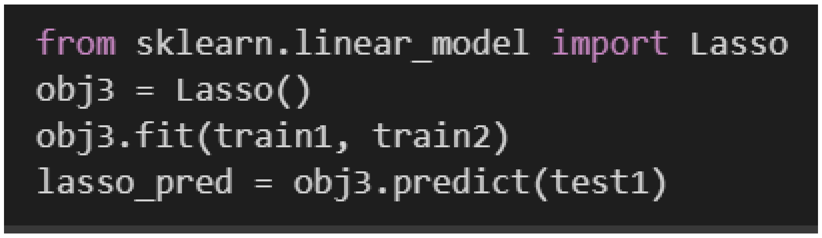
Decision Tree Regressor



```
from sklearn.tree import DecisionTreeRegressor
obj2 = DecisionTreeRegressor()
obj2.fit(train1, train2)
dec_pred = obj2.predict(test1)
```

The decision tree regressor is a model using which the data is broken down into smaller subsets and this makes the process more manageable and helps in incremental development of the decision tree. Using this regressor the regression baked problems are solved and this model is directly imported from the sklearn.tree module of the sklearn library and the two sets of training dataset are used for the training of the model. This trained model generates predictions using the one set of testing data.

Lasso model



```
from sklearn.linear_model import Lasso
obj3 = Lasso()
obj3.fit(train1, train2)
lasso_pred = obj3.predict(test1)
```

The lasso model is Least Absolute Shrinkage and Selection Operator, this is used for performing the feature selection and the regularization of the models. Using this model, the coefficients are regularized or shrinked in order to avoid the overfitting and make them suitable to work for different datasets. This model is imported from the sklearn.linear_model module and is directly fitted over the training data and generates predictions using the one set of testing data (test1).

Results

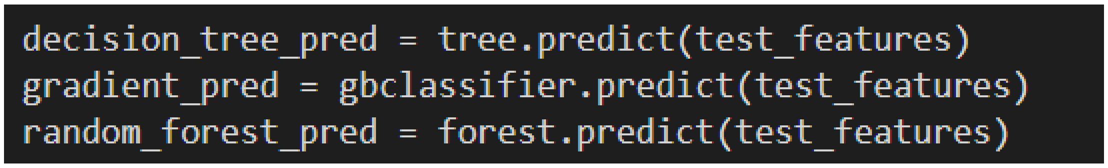
Classification Models

```
from sklearn.metrics import accuracy_scoreprint('accuracy of decision tree classifier is ',
accuracy_score(test2, decision_pred))
print('accuracy of random forest classifier', accuracy_score(test2, random_pred))
print('accuracy of gradient Boosting classifier', accuracy_score(test2, gb_pred))
```

```
accuracy of decision tree classifier is  0.776
accuracy of random forest classifier 0.776
accuracy of gradient boosting classifier  0.848
```

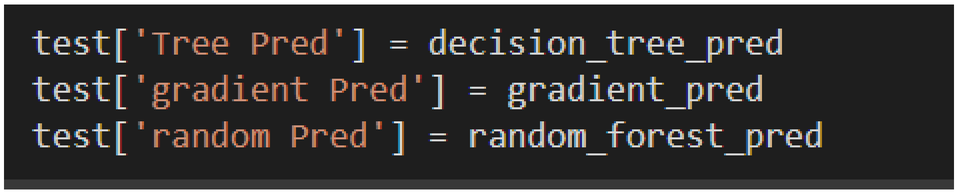
In order to evaluate the performance and working of the implemented classifier models, the accuracy metrics are calculated and for this the accuracy_score metric is used and thai metric is imported from the sklearn.metrics module of sklearn library. From all the classification models, the accuracy of the gradient boosting classifier is the highest (84 %) in comparison with the other two classifier models. This accuracy value clearly indicates that this model is highly efficient in making accurate predictions and is capable of generating more efficient results. In order to generate the accuracy of all these classifier models the another set of testing data and the generated predictions of each model are compared.

The predictions generated using the testing dataset.



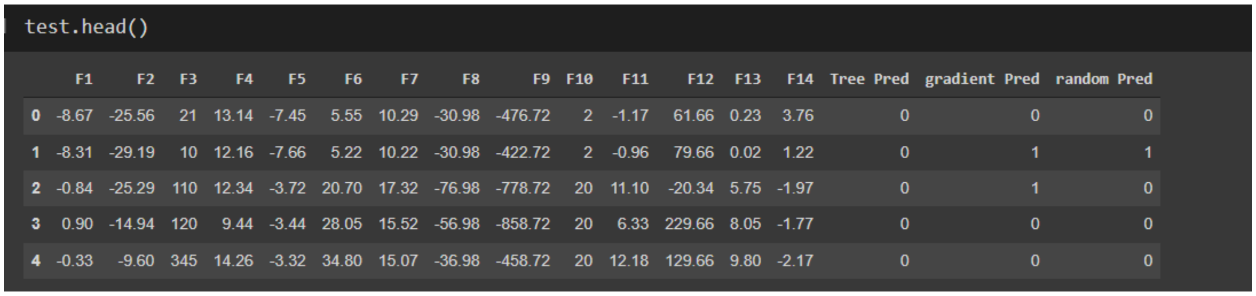
```
decision_tree_pred = tree.predict(test_features)
gradient_pred = gbclassifier.predict(test_features)
random_forest_pred = forest.predict(test_features)
```

The image above shows that using the testing dataset, the predictions are made by each classifier model. This training dataset was loaded and named as the test_features.



```
test['Tree Pred'] = decision_tree_pred
test['gradient Pred'] = gradient_pred
test['random Pred'] = random_forest_pred
```

The figure above shows that these generated predictions are added as new columns to the testing dataset and these columns contain the generated predictions as their values, these values are in the integer format (0 or 1).



```
test.head()
```

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	Tree Pred	gradient Pred	random Pred
0	-8.67	-25.56	21	13.14	-7.45	5.55	10.29	-30.98	-476.72	2	-1.17	61.66	0.23	3.76	0	0	0
1	-8.31	-29.19	10	12.16	-7.66	5.22	10.22	-30.98	-422.72	2	-0.96	79.66	0.02	1.22	0	1	1
2	-0.84	-25.29	110	12.34	-3.72	20.70	17.32	-76.98	-778.72	20	11.10	-20.34	5.75	-1.97	0	1	0
3	0.90	-14.94	120	9.44	-3.44	28.05	15.52	-56.98	-858.72	20	6.33	229.66	8.05	-1.77	0	0	0
4	-0.33	-9.60	345	14.26	-3.32	34.80	15.07	-36.98	-458.72	20	12.18	129.66	9.80	-2.17	0	0	0

The table above shows the new attributes of the testing dataset added. Using the head() method of python the above table is displayed.

Regression Models

from sklearn.metrics import r2_score

print('The r2 score of linear regression model using first dataset ', r2_score(linear_pred, test2))

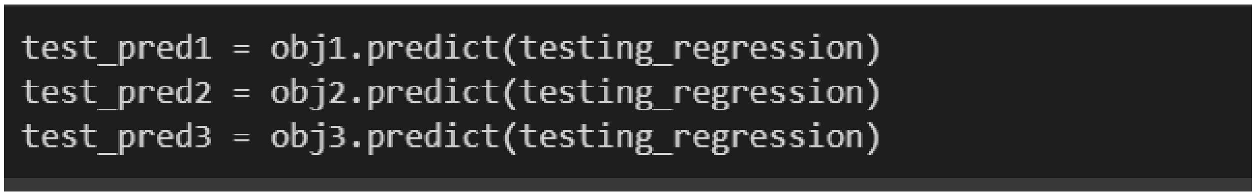
print('The r2 score of Decision Tree Regressor using first dataset ', r2_score(dec_pred, test2))

print('The r2 score of Lasso model using first dataset ', r2_score(lasso_pred, test2))

The r2 score of linear regression model 0.5510864054695737
The r2 score of Decision Tree Regressor 0.18611634019630086
The r2 score of Lasso model   0.5506512235381603

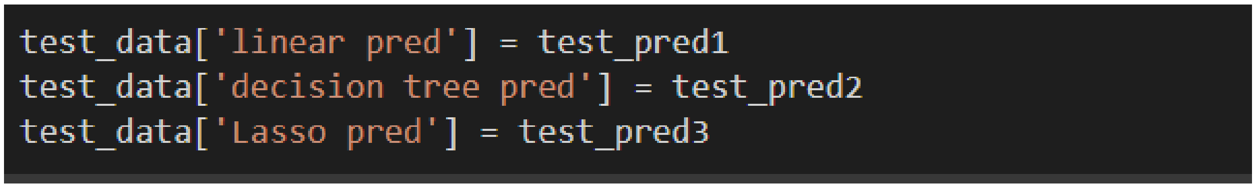
For evaluating the performance of the regression models, the r2_score metric is used. This metric is an important metric using which the performance of the regression based models is evaluated. This is also called the coefficient of determination and use for measuring the amount of variance present in the generated predictions. This metric is imported from the sklearn.metrics library of python and both the regression based models which are linear regression and lasso model are having same value of r2_ score which is approx 55 %, this shows the amount of variance present and the regression model which is having the least variance is the decision tree regressor. For generating the value of r2_score the generated predictions and another set of testing data is used.

The predictions generated using the testing dataset.



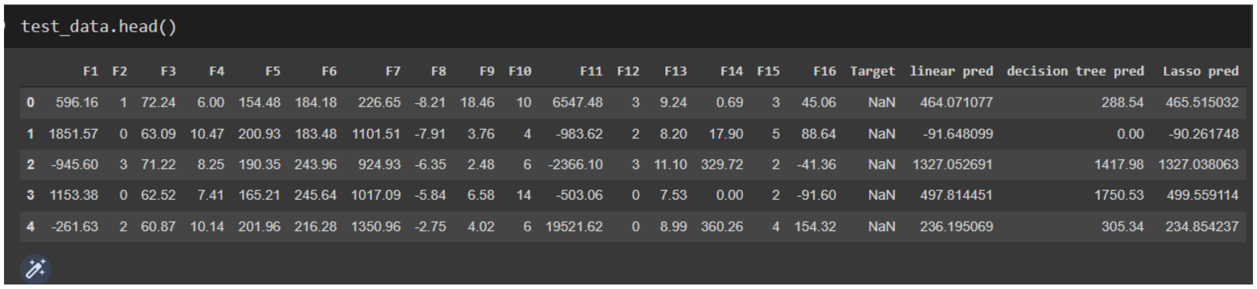
```
test_pred1 = obj1.predict(testing_regression)
test_pred2 = obj2.predict(testing_regression)
test_pred3 = obj3.predict(testing_regression)
```

The figure above shows the use of a testing dataset for generating the predictions of all the three implemented regression models.



```
test_data['linear pred'] = test_pred1
test_data['decision tree pred'] = test_pred2
test_data['Lasso pred'] = test_pred3
```

The figure above shows that new attributes are created and are added to the testing dataset. The value of these newly added attributes are in the float format and some of the values are in negative.



```
test_data.head()
```

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	Target	linear pred	decision tree pred	Lasso pred
0	596.16	1	72.24	6.00	154.48	184.18	226.65	-8.21	18.46	10	6547.48	3	9.24	0.69	3	45.06	NaN	464.071077	288.54	465.515032
1	1851.57	0	63.09	10.47	200.93	183.48	1101.51	-7.91	3.76	4	-983.62	2	8.20	17.90	5	88.64	NaN	-91.648099	0.00	-90.261748
2	-945.60	3	71.22	8.25	190.35	243.96	924.93	-6.35	2.48	6	-2366.10	3	11.10	329.72	2	-41.36	NaN	1327.052691	1417.98	1327.038063
3	1153.38	0	62.52	7.41	165.21	245.64	1017.09	-5.84	6.58	14	-503.06	0	7.53	0.00	2	-91.60	NaN	497.814451	1750.53	499.559114
4	-261.63	2	60.87	10.14	201.96	216.28	1350.96	-2.75	4.02	6	19521.62	0	8.99	360.26	4	154.32	NaN	236.195069	305.34	234.854237

The table above shows the implemented models which are added to the training dataset as the new attributes. This table is displayed using the .head() method of python.

Interpretation of the results

The generated results of both the classification and regression based models shows that the classification models are having higher accuracy and the regression models are having less variance. But the best performing classifier model is the gradient boosting classifier having the highest accuracy and the other two models are also working satisfactorily. On the other hand from the implemented regression models, the decision tree regressor model is having the least variance present in the generated predictions and the other two models are having approx half of the variance in the generated predictions. By analyzing the results of all the models it can be said that the regression tasks were not implemented effectively because the selected data for the process was not very effective in generating the results and much more refined data was required and for the regression processes lareg datasets are required to be processed.

References

- Kim, Y., Hao, J., Mallavarapu, T., Park, J. and Kang, M., 2019. Hi-LASSO: High-Dimensional LASSO. IEEE Access, [online] 7, pp.44562-44573. Available at: <https://ieeexplore.ieee.org/document/8684195>.
- Mathur, A. and Foody, G., 2008. Multiclass and Binary SVM Classification: Implications for Training and Classification Users. *IEEE Geoscience and Remote Sensing Letters*, [online] 5(2), pp.241-245. Available at: <https://ieeexplore.ieee.org/document/4460898>.
- Wu, J., Huang, L. and Pan, X., 2010. A Novel Bayesian Additive Regression Trees Ensemble Model Based on Linear Regression and Nonlinear Regression for Torrential Rain Forecasting. 2010 Third International Joint Conference on Computational Science and Optimization, [online] Available at: <https://ieeexplore.ieee.org/document/5532940>.

- Yang, M., Wu, K. and Hsieh, J., 2007. Mountain C-Regressions in Comparing Fuzzy C-Regressions. *2007 IEEE International Fuzzy Systems Conference*, [online] Available at: <https://ieeexplore.ieee.org/document/4295366>.