CT4101-Machine Learning Assignement-3

Name : Saikrishna Javvadi **Student-ID** : 20236648 **Class** : MSc in CS– Data Analytics

Package of choice and it's features:

For my assignment, I will use the Scikit-learn ML Framework, because Scikit-learn offers a good range of useful tools for metrics functions, data splitting, data manipulation, generation of artificial datasets, and so on. Scikit-learn also has strong documentation and a consistent ,clean API, so we are lost anywhere while programming because of the immense online community support, it is also easier to get help on the internet. Another primary benefit of this kit is that it is easy to navigate and a good option for simpler tasks of data analysis[1]. Some of the main features of this package are:
1) NumPy, SciPy, and matplotlib are based on scikit-learn, so it fits well with all these libraries. 2) Free, commercially available, source. 3) In different ways, Scikit-learn is reusable and is open to all. 4) By using scikit-learn, a range of pre-processing measures such as taking care of missing data, encoding categorical data and scaling of features could be easily accomplished. 5) Various model selection steps, such as dividing the dataset into training and test sets, tuning an estimator's hyperparameters, etc., could easily be achieved using this kit. 6) Scikit-learn implements a variety of non-neural networks based algorithms that are widely used in data science, such as classification, regression, and clustering. 7) The number of attributes in data can be reduced for further visualization, summarization, using Dimensionality reduction in scikit-learn.[1]

Data Pre-processing:

Firstly, the data was loaded from the given text file into a pandas dataframe and I assigned names to dataframe columns accordingly. Later I rearranged the column names to have the dependent variable at the end for my ease of use. It was found that there were duplicates in the data, so I dropped the duplicates keeping only their first occurrences. Then, the dataframe is converted to numpy arrays X_data and Y_data which contain independent variables and dependent variable respectively. Using the train_test_split class of sklearn.model_selection , the data is split into training and test sets. Then, using the StandardScaler class of sklearn.preprocessing , a feature scaling step was done to reduce the impact of a single variable on Bayesian distance which didn't help much in increasing the performance of the model. However the scaled data is only used for Linear regression task, as we don't have to scale data for Randomforest regressor, which is explained further while describing the algorithm.

Algorithms used & their Description:

The two algorithms that I used for my regression task are Multiple Linear Regression and Random forest regression. Both the algorithms are described below :

Multiple Linear Regression:

By fitting a line to the observed data, Linear regression models are used to explain relations between variables. Regression helps us to predict the shift of a dependent variable as the independent variable(s) changes. Multiple Linear regression is designed to create regressions on models with a single dependent variable and multiple independent variables, unlike simple linear regression which deals with single dependent variable and a single independent

variable. The equation of multiple linear regression takes the form: **y = B_0 + B_1 * x_1 + … + B_n * x_n** where B_1,B_2…,B_n are the coefficients of the attributes x_1,x_2,..x_n and B_0 is the bias/constant term. We need to find a hypothesis that minimizes the sum of squared errors to find all of the B terms in multiple regression using gradient descent or any other optimization algorithm. Since there are more parameters in this model than in simple linear regression, when constructing the equation, more care should be taken. Adding more terms would inherently increase the fit for the data, but there may be no physical significance for the additional terms. This is dangerous because it leads to a model that matches that knowledge, but really does not mean anything useful. More features also increase the risk of overfitting the model, leading to potentially bad outcomes when actually predicting values.[2]

Random Forest Regressor:

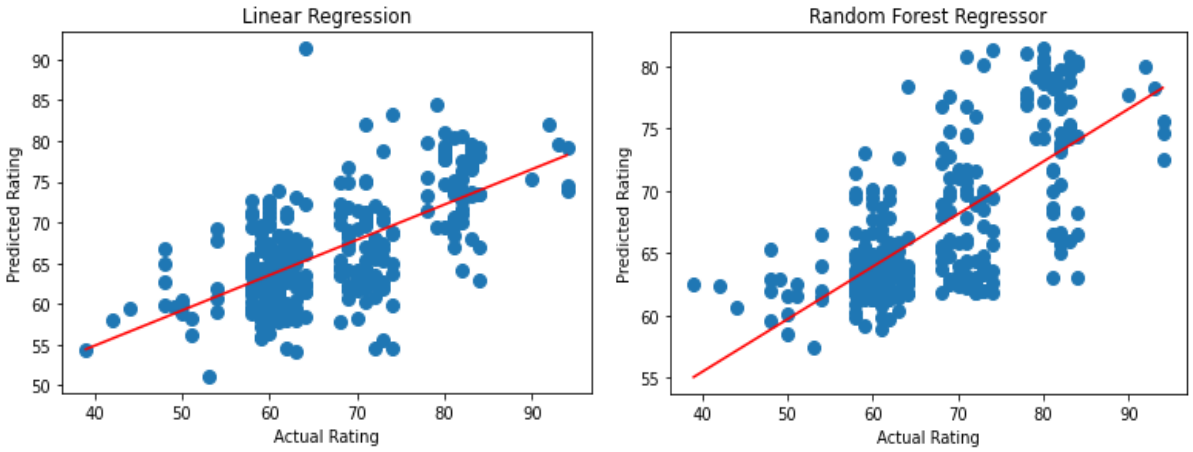
A nonparametric supervised learning methodology used for classification and regression is Decision Trees (DTs). The objective is to construct a model that predicts the value of a target variable by learning basic rules of judgment inferred from the data characteristics. Predictions from a decision tree regression or random forest regression model are resulted from successive splits of the data through the different nodes of the tree, hence we can't construct a equation like we do for other models. For this very reason we don't have to do feature scaling for these algorithms. Through the use of multiple decision trees and a methodology called Bootstrap Aggregation, an ensemble technique like Random forest can be built which is capable of performing both regression and classification tasks. Ensemble learning is when you take multiple algorithms or the same algorithm multiple times and you put them together to make something much more powerful than the original and Random forest is a version of ensemble learning . Bagging requires training each decision tree on a different data sample in the Random Forest process, where sampling is done with replacement. The fundamental concept behind this is to combine multiple decision trees rather than relying on individual decision trees in deciding the final output.

The basic steps involved in using the random forest algorithm are the following:

1)From the dataset, select N random records. 2) Based on these N records, create a decision tree. 3)In your algorithm, pick the number of trees you would like and repeat steps 1 and 2. 4)In the case of a regression problem, each tree in the forest predicts a Y value for a new record (output). By taking the average of all the values expected by all the trees in the forest, the final value can be determined.[3][4][5]

Implementing models of the above chosen algorithms:

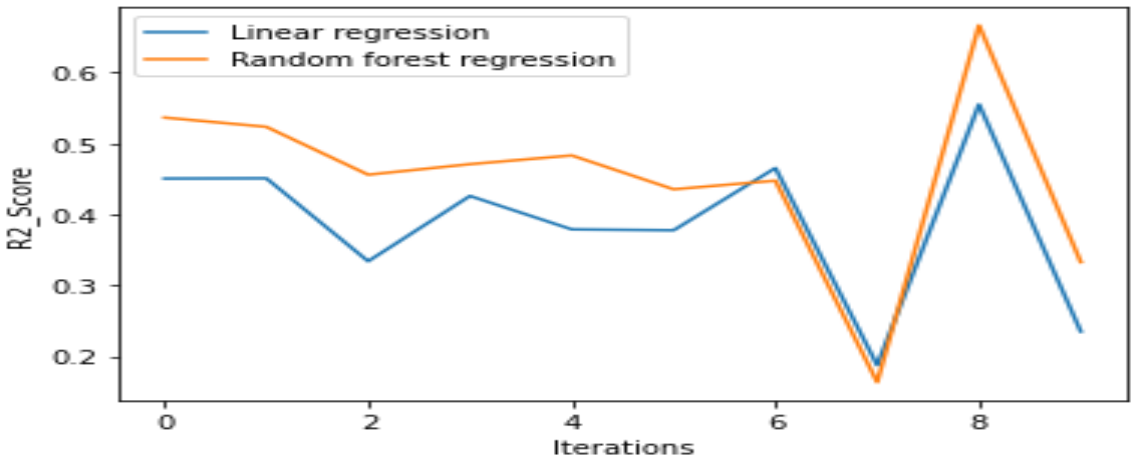
The above selected algorithms were easy to implement for the given data because of scikit-learn's already available classes/built-in methods. Firstly the pre-processed training data was fit into the linear regression model using the LinearRegression class of sklearn.linear_model library. Similary, using the RandomForestRegressor of sklearn.ensemble library , the training data was fit into the Random forest regression model. To tune the hyperparameters like n_estimators and max_depth for the RandomforestRegressor, I used gridSearchCV of the sklearn.model_selection API[5]. We can see how the predicted rating is varying from the actual rating for the two algorithms in the figure below. Since the r2_score for these initially models was less(around 0.38 to 0.47) , I did feature selection using backward elimination to find the significance of variables using statsmodels.api to fit the models only with relevant attributes.



The steps used/involved while doing feature selection using backward elimination were:
1) Select a Significance level for the model, usually SL=0.05. 2)Fit the model with all variables. 3)Consider Predictor with highest p-value. If p-value>SL, go to step4, else our model is ready. 4)Remove the Predictor and fit the model without this variable, repeat.[6]
However, even this step didn't help me much to increase the performance of the model.

Monitoring for Possible Overfitting and underfitting:

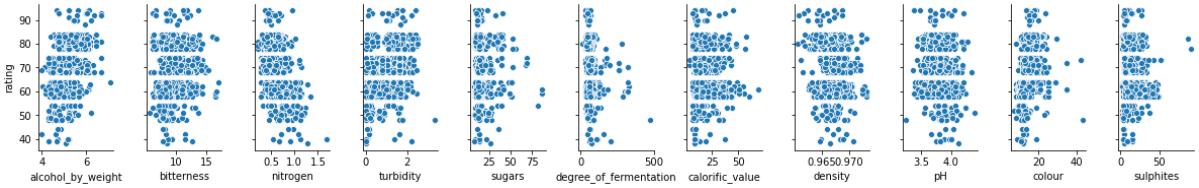
Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms. Overfitting refers to a model that models the training data too well. Nonparametric and nonlinear models like decision trees and Random Forest that have more versatility in learning a target function are more likely to overfit. Underfitting refers to a model which can neither model nor generalize the training data to new data[7]. Since the r2_score for the training and test sets doesn't vary much for the Linear regression model, there wasn't much to be done here. But before tuning the hyperparameters, the Randomforest regressor algorithm was overfitting the training data. Later, after the usage of gridsearchCV to tune the hyperparameters the overfitting of the algorithm to training data was considerably reduced. The k-fold cross validation is the most common resampling technique which I used to avoid overfitting. It enables us to train and test our model on different sub-sets of training data k-times and to construct an estimation of the performance of a machine learning model on unseen data[8]. We can observe the variance of r2_score with respect to each fold for the given dataset in the figure below.



Interpreting the performance of two Regression models used:

I choose r2_score and RMSE as my two metrics to evaluate the performance of my regression models. The most common understanding of r2-score is how well the regression model fits the observed data. Heuristically , RMSE can be interpreted as some form of (normalized) distance between the predicted values vector and the actual/observed values vector. R-squared is a relative fit measure, while RMSE is an absolute fit measure. Generally, a higher r2-score and a lower RMSE indicates a better fit for the model[9]. For our models, we can observe that Linear regression model results in an r2_score of 0.38-0.45 and an RMSE of 7.5-7.9, whereas the random forest regressor results in a r2_score of 0.42-0.54 and RMSE of 7.1-7.6.

Even though performances of two models isn't much different, we can observe that random forest regressor does a slightly better job for the given data. Also, From the below pairplot which plotted for each independent attribute with respect to the dependent attribute(rating) using Seaborn library's pairplot , we can observe how the dependent variable is varying with respect to each of the independent variable. I observed that there was no linear relationship between any of the independent attributes with the target attribute, there are only clusters of points grouped together that can be observed below, which might be a contributing factor to the not so good performance of the models.



References:

1) https://scikit-learn.org/stable/
2) https://towardsdatascience.com/understanding-multiple-regression-249b16bde83e
3) https://stackabuse.com/random-forest-algorithm-with-python-and-scikit-learn/
4)https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
5) https://medium.com/datadriveninvestor/random-forest-regression-9871bc9a25eb
6) https://medium.com/@mayankshah1607/machine-learning-feature-selection-with-backward-elimination-955894654026
7) https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
8) https://towardsdatascience.com/why-and-how-to-cross-validate-a-model-d6424b45261f
9) https://www.theanalysisfactor.com/assessing-the-fit-of-regression-models

Appendix : Code

```
import pandas as pd
import numpy as np
data = pd.read_csv('C:\\Users\\javva\\Downloads\\beer_ratings.txt', sep="\t", header=None)
data.columns= ['alcohol_by_weight', 'rating', 'bitterness', 'nitrogen', 'turbidity', 'sugars',
'degree_of_fermentation', 'calorific_value', 'density', 'pH', 'colour', 'sulphites']
#Rearranging columns
data = data[['alcohol_by_weight', 'bitterness', 'nitrogen', 'turbidity', 'sugars',
'degree_of_fermentation', 'calorific_value', 'density', 'pH', 'colour', 'sulphites', 'rating']]
print(data.columns)
data.head()
data.info()
data.describe()

#Checking for duplicates
duplicate = data[data.duplicated()]
print("Duplicate Rows :")
duplicate

#dropping duplicates
data.drop_duplicates(keep='first',inplace=True)
print(len(data))

import seaborn as sns
#sns.pairplot(data)
#plotting all the independent variables against the dependent variable
sns.pairplot(data,x_vars=["alcohol_by_weight", "bitterness", "nitrogen", "turbidity", "sugars",
"degree_of_fermentation", "calorific_value", "density", "pH", "colour",
"sulphites"],y_vars=["rating"],aspect=0.6)

#Splitting the data
from sklearn.model_selection import train_test_split
X_data,Y_data = data.iloc[:,:-1].values, data.iloc[:,-1].values
print(X_data.shape)
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, train_size = 2/3, random_state = 0)
X_train
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

#Implementing Linear regression and RandomForest Regression models
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import metrics

model = LinearRegression()
```

```
model.fit(X_train_scaled,Y_train)
regressor_rf = RandomForestRegressor(n_estimators = 100, random_state = 0,max_depth=4)
regressor_rf.fit(X_train,Y_train)

#Predicting the values using the implemented models
y_train_pred_lr=model.predict(X_train_scaled)
y_test_pred_lr=model.predict(X_test_scaled)
y_train_pred_RFR=regressor_rf.predict(X_train)
y_test_pred_RFR=regressor_rf.predict(X_test)

#Evaluating the performance of the two regression models
print('Linear Regression R2_Score for Training Data=>', model.score(X_train_scaled,Y_train))
print('Linear Regression R2_Score for Test Data=>',model.score(X_test_scaled,Y_test))
print('Linear Regression RMSE for Train
data=>',np.sqrt(mean_squared_error(Y_train,y_train_pred_lr)))
print('Linear Regression RMSE for Test data=>',np.sqrt(mean_squared_error(Y_test,y_test_pred_lr)))
print('RandomForest Regressor R2_Score for Training Data=>', regressor_rf.score(X_train,Y_train))
print('RandomForest Regressor R2_Score for Test Data=>',regressor_rf.score(X_test,Y_test))
print('RandomForest Regressor RMSE for Train
data=>',np.sqrt(mean_squared_error(Y_train,y_train_pred_RFR)))
print('RandomForest Regressor RMSE for Test
data=>',np.sqrt(mean_squared_error(Y_test,y_test_pred_RFR)))

#Using GridSearchCV to tune the hyperparameters for Random Forest regressor
from sklearn.model_selection import GridSearchCV
gsc = GridSearchCV(
      estimator=RandomForestRegressor(),
      param_grid={
         'max_depth': range(3,9),
         'n_estimators': (10, 50,75, 100, 1000),
      },
      cv=5, scoring='neg_mean_squared_error', verbose=0,n_jobs=-1)
grid_result = gsc.fit(X_data, Y_data)
best_params = grid_result.best_params_
best_params

#Implementing k-fold cross-validation
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, cross_val_predict
import numpy as np
import matplotlib.pyplot as plt

model = LinearRegression()
regressor_rf = RandomForestRegressor(n_estimators = 100,random_state = 0,max_depth=4)

folds = KFold(n_splits = 5, shuffle = True,random_state=0)
cv_r2_scores_lr = cross_val_score(model, X_data, Y_data, cv=folds,scoring='r2')
print("Linear Regression Mean k-Fold R Squared: {}".format(np.mean(cv_r2_scores_lr)))
scores = cross_val_score(model, X_data, Y_data, scoring='neg_mean_squared_error', cv=folds)
print("Linear Regression Mean k-Fold RMSE: {}".format(np.sqrt(abs(np.mean(scores)))))
```

```
cv_r2_scores_rf = cross_val_score(regressor_rf, X_data, Y_data, cv=folds,scoring='r2')
print("RandomForest Regressor Mean k-Fold R Squared: {}".format(np.mean(cv_r2_scores_rf)))
scores = cross_val_score(regressor_rf, X_data, Y_data, scoring='neg_mean_squared_error', cv=folds)
print("RandomForest Regressor Mean k-Fold RMSE: {}".format(np.sqrt(abs(np.mean(scores)))))

#Plotting the R2_score vs Iterations graph for the two algorithms
plt.plot(range(10), cv_r2_scores_lr, label ="Linear regression")
plt.plot(range(10), cv_r2_scores_rf, label ="RandomForest regression")
plt.xlabel("Iterations")
plt.ylabel("R2_Score")
plt.legend()
plt.show()

#Plotting the graph of actual rating to the predicted rating for each algorithm
import numpy as np
import matplotlib.pyplot as plt
plt.figure()
plt.plot(Y_test, y_test_pred_lr, 'o', markersize = 7 )
plt.title('Linear Regression')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.plot(np.unique(Y_test), np.poly1d(np.polyfit(Y_test, y_test_pred_lr, 1))(np.unique(Y_test)), color
= 'green')
plt.show()

plt.figure()
plt.plot(Y_test, y_test_pred_RFR, 'o', markersize = 7 )
plt.title('RandomForest Regressor')
plt.xlabel('Actual Rating')
plt.ylabel('Predicted Rating')
plt.plot(np.unique(Y_test), np.poly1d(np.polyfit(Y_test, y_test_pred_RFR, 1))(np.unique(Y_test)),
color = 'green')
plt.show()

#Using backward elimination to find significance of the variables
X_train = np.append (arr=np.ones([X_train.shape[0],1]).astype(int), values = X_train, axis = 1)
X_train.shape
import statsmodels.api as sm
X_opt = [0,1,2,3,4,5,6,7,8,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())
X_opt = [0,1,2,3,5,6,7,8,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())
X_opt = [0,1,2,3,6,7,8,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())
X_opt = [0,1,3,6,7,8,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())
```

```
X_opt = [0,1,3,6,7,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())
X_opt = [0,1,3,6,9,10,11]
regressor = sm.OLS(Y_train, X_train[:,X_opt]).fit()
print(regressor.summary())

#Splitting the data into training and test sets using only significant features determined from the
previous step
X_train, X_test, Y_train, Y_test = train_test_split(X_data[:,[0,2,5,8,9,10]], Y_data, train_size = 2/3,
random_state = 0)
print(X_train.shape)
#Feature Scaling
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

#Repeating the above steps again
model = LinearRegression()
model.fit(X_train_scaled,Y_train)
regressor_rf = RandomForestRegressor(n_estimators = 100, random_state = 0,max_depth=4)
regressor_rf.fit(X_train,Y_train)

y_train_pred_lr=model.predict(X_train_scaled)
y_test_pred_lr=model.predict(X_test_scaled)
y_train_pred_RFR=regressor_rf.predict(X_train)
y_test_pred_RFR=regressor_rf.predict(X_test)

print('Linear Regression R2_Score for Training Data=>', model.score(X_train_scaled,Y_train))
print('Linear Regression R2_Score for Test Data=>',model.score(X_test_scaled,Y_test))
print('Linear Regression RMSE for Train
data=>',np.sqrt(mean_squared_error(Y_train,y_train_pred_lr)))
print('Linear Regression RMSE for Test data=>',np.sqrt(mean_squared_error(Y_test,y_test_pred_lr)))
print('RandomForest Regressor R2_Score for Training Data=>', regressor_rf.score(X_train,Y_train))
print('RandomForest Regressor R2_Score for Test Data=>',regressor_rf.score(X_test,Y_test))
print('RandomForest Regressor RMSE for Train
data=>',np.sqrt(mean_squared_error(Y_train,y_train_pred_RFR)))
print('RandomForest Regressor RMSE for Test
data=>',np.sqrt(mean_squared_error(Y_test,y_test_pred_RFR)))

folds = KFold(n_splits = 10, shuffle = True,random_state=0)
cv_r2_scores_lr = cross_val_score(model, X_data[:,[0,2,5,8,9,10]], Y_data, cv=folds,scoring='r2')
print("Mean 5-Fold R Squared: {}".format(np.mean(cv_r2_scores_lr)))
scores = cross_val_score(model, X_data, Y_data, scoring='neg_mean_squared_error', cv=folds)
print("Mean 5-Fold RMSE: {}".format(np.sqrt(abs(np.mean(scores)))))

cv_r2_scores_rf = cross_val_score(regressor_rf, X_data[:,[0,2,5,8,9,10]], Y_data,
cv=folds,scoring='r2')
print("Mean 5-Fold R Squared: {}".format(np.mean(cv_r2_scores_rf)))
scores = cross_val_score(regressor_rf, X_data, Y_data, scoring='neg_mean_squared_error', cv=folds)
print("Mean 5-Fold RMSE: {}".format(np.sqrt(abs(np.mean(scores)))))
```