ML LAB 8

Name: Soumyadeep Ganguly

Reg No: 24MDT0082

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [5]:
```
df = pd.read_csv('Book1.csv')
df.head()
df = df.drop(['furnishingstatus'], axis = 1)
df.head()
```

Out[5]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

In [6]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled = scaler.fit_transform(df)
df_scaled = pd.DataFrame(scaled, columns = df.columns)
df_scaled.head()
```

Out[6]:

	price	area	bedrooms	bathrooms	stories	parking
0	1.000000	0.356777	0.50	0.333333	0.666667	0.666667
1	0.880096	0.469597	0.50	1.000000	1.000000	1.000000
2	0.880096	0.542857	0.25	0.333333	0.333333	0.666667
3	0.876099	0.362637	0.50	0.333333	0.333333	1.000000
4	0.784173	0.356777	0.50	0.000000	0.333333	0.666667

In [7]:
```
from sklearn.model_selection import train_test_split

X = df_scaled.drop(['price'], axis=1)
y = df_scaled['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state= 42)
```

In [8]:
```
from sklearn.tree import DecisionTreeRegressor

DTR = DecisionTreeRegressor()
DTR.fit(X_train, y_train)
```

Out[8]:

▾ DecisionTreeRegressor

```
DecisionTreeRegressor()
```

In [9]:
```
y_pred = DTR.predict(X_test)
y_pred
```

Out[9]:
```
array([0.12070344, 0.50439648, 0.46842526, 0.32054357, 0.24060751,
       0.11270983, 0.16067146, 0.46842526, 0.18864908, 0.11270983,
       0.48041567, 0.18465228, 0.01278977, 0.24060751, 0.87609912,
       0.07673861, 0.03277378, 0.23820943, 0.16067146, 0.88009592,
       0.068745  , 0.28856914, 0.21422862, 0.24060751, 0.88009592,
       0.24060751, 0.23820943, 0.1518785 , 0.50439648, 0.17665867,
       0.01678657, 0.58673062, 0.27364775, 0.10871303, 0.54436451,
       0.03277378, 0.46842526, 0.58673062, 0.18465228, 0.16866507,
       0.3804956 , 0.28856914, 0.068745  , 0.10871303, 0.78417266,
       0.17665867, 0.2246203 , 0.88009592, 0.03277378, 0.14468425])
```

In [10]:
```
from sklearn.metrics import mean_squared_error

DTR_Error = mean_squared_error(y_test, y_pred)
print(f'Error is: {DTR_Error}')
```

```
Error is: 0.07675530700761357
```

Gradient Boosting Regressor

In [11]:
```
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor(learning_rate=0.01, n_estimators=100, max_depth = 3, random_state=0)
GBR.fit(X_train, y_train)
```

Out[11]:
```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.01, random_state=0)
```

In [12]:
```
y_pred = GBR.predict(X_test)
y_pred
```

Out[12]:
```
array([0.17414174, 0.4752913 , 0.17071984, 0.23444211, 0.19341953,
       0.20503384, 0.15387844, 0.18093334, 0.23547499, 0.20453393,
       0.15984301, 0.15009756, 0.12134007, 0.15984301, 0.40508986,
       0.18893   , 0.11118068, 0.18085147, 0.21885721, 0.42330002,
       0.11107113, 0.23444211, 0.18048463, 0.14834259, 0.42330002,
       0.21979456, 0.16655274, 0.19840865, 0.42433569, 0.18048463,
       0.12134007, 0.23172445, 0.21271041, 0.12516125, 0.23494203,
       0.1991435 , 0.22773238, 0.23222437, 0.17213165, 0.18093334,
       0.15285543, 0.18093334, 0.15200926, 0.14644912, 0.277752  ,
       0.23494203, 0.25775041, 0.42330002, 0.1991435 , 0.17370852])
```

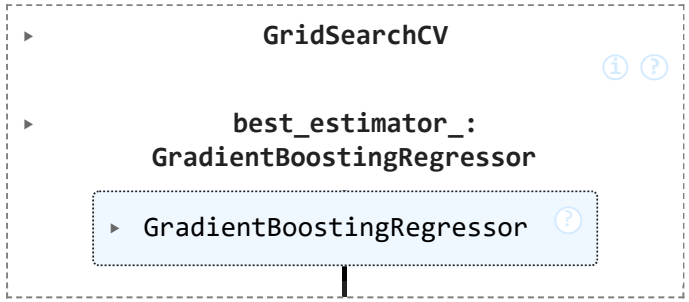
In [13]:
```
GBR_Error = mean_squared_error(y_pred, y_test)
print(f'Error is: {GBR_Error}')
```

```
Error is: 0.026793767598257433
```

In [14]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
 'n_estimators':[ i for i in range(50, 501, 50)],
 'learning_rate': list(np.linspace(0,1, 10)),
 'max_depth': list(np.random.randint(1,10, 10))
}
base_model = GradientBoostingRegressor(random_state=0)
```

In [15]:
```
gs = GridSearchCV(estimator = base_model, param_grid = param_grid,cv=5, n_jobs = -1)
gs.fit(X_train, y_train)
```

Out[15]:



In [16]:
```
print(f'Best parameters for "Gradient Boosting Regressor": {gs.best_params_}')
print(f'Best Accuracy: {gs.best_score_}')
```

```
Best parameters for "Gradient Boosting Regressor": {'learning_rate': np.float64(0.1111111111111111), 'max_depth': np.int32(1),
'n_estimators': 200}
Best Accuracy: 0.24953014819241837
```

liver_patient.csv

In [17]:
```
df2 = pd.read_csv("liver_patient.csv")
df2 = df2.drop(['Age', 'Gender'], axis = 1)
df2.head()
```

Out[17]:

	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
0	0.7	0.1	187	16	18	6.8	3.3
1	10.9	5.5	699	64	100	7.5	3.2
2	7.3	4.1	490	60	68	7.0	3.3
3	1.0	0.4	182	14	20	6.8	3.4
4	3.9	2.0	195	27	59	7.3	2.4

In [18]:
```
scaler2 = MinMaxScaler(feature_range=(0,1))
scaled2 = scaler2.fit_transform(df2)
df2_scaled = pd.DataFrame(scaled2, columns = df2.columns)
df2_scaled.head()
```

Out[18]:

	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
0	0.004021	0.000000	0.060576	0.003015	0.001626	0.594203	0.521739
1	0.140751	0.275510	0.310699	0.027136	0.018296	0.695652	0.500000
2	0.092493	0.204082	0.208598	0.025126	0.011791	0.623188	0.521739
3	0.008043	0.015306	0.058134	0.002010	0.002033	0.594203	0.543478
4	0.046917	0.096939	0.064485	0.008543	0.009961	0.666667	0.326087

In [19]:
```
X2 = df2_scaled.drop(['liver_disease'], axis = 1)
y2 = df2_scaled['liver_disease']

X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2, test_size = 0.2, random_state=0)
```

In [20]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
DTC = DecisionTreeClassifier()
DTC.fit(X_train2, y_train2)
```

Out[20]:
```
DecisionTreeClassifier()
```

In [21]:
```
y_pred2 = DTC.predict(X_test2)
```

In [22]:
```
DTC_score = accuracy_score(y_test2, y_pred2)
print(f'Error is: {DTC_score}')
```

```
Error is: 0.5811965811965812
```

In [23]:
```
from sklearn.ensemble import GradientBoostingClassifier
GBC = GradientBoostingClassifier(learning_rate=0.01, n_estimators=100, max_depth = 3, random_state=0)
GBC.fit(X_train2, y_train2)
```

Out[23]:
```
GradientBoostingClassifier(learning_rate=0.01, random_state=0)
```

Q2

In [24]:
```
param_grid2 = {
 'n_estimators':[ i for i in range(50, 501, 50)],
 'learning_rate': list(np.linspace(0,1, 10)),
 'max_depth': list(np.random.randint(1,10, 10))
}
base_model2 = GradientBoostingClassifier(random_state=0)
gs2 = GridSearchCV(estimator = base_model2, param_grid = param_grid2,cv=5, n_jobs = -1)
gs2.fit(X_train2, y_train2)
```

Out[24]:

GridSearchCV

best_estimator_: GradientBoostingClassifier

GradientBoostingClassifier

In [25]:
```
print(f'Best parameters for "Gradient Boosting Classifier": {gs2.best_params_}')
print(f'Best Accuracy: {gs2.best_score_}')
```

```
Best parameters for "Gradient Boosting Classifier": {'learning_rate': np.float64(0.1111111111111111), 'max_depth': np.int32(1),
'n_estimators': 50}
Best Accuracy: 0.7424388011896592
```

Reguarization techniques: Ridge and lasso regression.

Now we will try to look at ridge and lasso regression which are again regularization techniques used to minimize the variance or reduce overfitting of data. The lasso regression also kind of helps to know the best features in the modeling. Because it will take some coefficients of

the model which are not that relevant to zero

Q3

In [32]:
```
from sklearn.linear_model import RidgeCV, LassoCV
```

In [26]:
```
df3 = df.copy()
df3.head()

scaler3 = MinMaxScaler()
```

In [28]:
```
df3.head(2)
```

Out[28]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3

In [29]:
```
data = scaler3.fit_transform(df)
df3 = pd.DataFrame(data=data, columns=df3.columns)
df3.head(2)
```

Out[29]:

	price	area	bedrooms	bathrooms	stories	parking
0	1.000000	0.356777	0.5	0.333333	0.666667	0.666667
1	0.880096	0.469597	0.5	1.000000	1.000000	1.000000

In [30]:
```
X = df3.iloc[:,1:].values
y = df3.iloc[:,0].values
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

In [31]:
```
alpha_values = np.logspace(-2,4,100)
alpha_values
```

Out[31]:
```
array([1.00000000e-02, 1.14975700e-02, 1.32194115e-02, 1.51991108e-02,
       1.74752840e-02, 2.00923300e-02, 2.31012970e-02, 2.65608778e-02,
       3.05385551e-02, 3.51119173e-02, 4.03701726e-02, 4.64158883e-02,
       5.33669923e-02, 6.13590727e-02, 7.05480231e-02, 8.11130831e-02,
       9.32603347e-02, 1.07226722e-01, 1.23284674e-01, 1.41747416e-01,
       1.62975083e-01, 1.87381742e-01, 2.15443469e-01, 2.47707636e-01,
       2.84803587e-01, 3.27454916e-01, 3.76493581e-01, 4.32876128e-01,
       4.97702356e-01, 5.72236766e-01, 6.57933225e-01, 7.56463328e-01,
       8.69749003e-01, 1.00000000e+00, 1.14975700e+00, 1.32194115e+00,
       1.51991108e+00, 1.74752840e+00, 2.00923300e+00, 2.31012970e+00,
       2.65608778e+00, 3.05385551e+00, 3.51119173e+00, 4.03701726e+00,
       4.64158883e+00, 5.33669923e+00, 6.13590727e+00, 7.05480231e+00,
       8.11130831e+00, 9.32603347e+00, 1.07226722e+01, 1.23284674e+01,
       1.41747416e+01, 1.62975083e+01, 1.87381742e+01, 2.15443469e+01,
       2.47707636e+01, 2.84803587e+01, 3.27454916e+01, 3.76493581e+01,
       4.32876128e+01, 4.97702356e+01, 5.72236766e+01, 6.57933225e+01,
       7.56463328e+01, 8.69749003e+01, 1.00000000e+02, 1.14975700e+02,
       1.32194115e+02, 1.51991108e+02, 1.74752840e+02, 2.00923300e+02,
       2.31012970e+02, 2.65608778e+02, 3.05385551e+02, 3.51119173e+02,
       4.03701726e+02, 4.64158883e+02, 5.33669923e+02, 6.13590727e+02,
       7.05480231e+02, 8.11130831e+02, 9.32603347e+02, 1.07226722e+03,
       1.23284674e+03, 1.41747416e+03, 1.62975083e+03, 1.87381742e+03,
       2.15443469e+03, 2.47707636e+03, 2.84803587e+03, 3.27454916e+03,
       3.76493581e+03, 4.32876128e+03, 4.97702356e+03, 5.72236766e+03,
       6.57933225e+03, 7.56463328e+03, 8.69749003e+03, 1.00000000e+04])
```

In [33]:
```
ridge_cv = RidgeCV(alphas=alpha_values, store_cv_values=True)
ridge_cv.fit(X_train,y_train)
```

```
e:\VIT Study Materials\SEM 2\Data Mining and ML\LAB\.venv\Lib\site-packages\sklearn\linear_model\_ridge.py:2375: FutureWarning:
'store_cv_values' is deprecated in version 1.5 and will be removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
```

Out[33]:

RidgeCV

```
RidgeCV(alphas=array([1.00000000e-02, 1.14975700e-02, 1.32194115e-02, 1.51991108e-02,
       1.74752840e-02, 2.00923300e-02, 2.31012970e-02, 2.65608778e-02,
       3.05385551e-02, 3.51119173e-02, 4.03701726e-02, 4.64158883e-02,
       5.33669923e-02, 6.13590727e-02, 7.05480231e-02, 8.11130831e-02,
       9.32603347e-02, 1.07226722e-01, 1.23284674e-01, 1.41747416e-01,
       1.62975083e-01, 1.87381742e-0...
       4.03701726e+02, 4.64158883e+02, 5.33669923e+02, 6.13590727e+02,
       7.05480231e+02, 8.11130831e+02, 9.32603347e+02, 1.07226722e+03,
       1.23284674e+03, 1.41747416e+03, 1.62975083e+03, 1.87381742e+03,
```

In [34]:
```
ridge_pred = ridge_cv.predict(X_test)
```

In [35]:
```
ridge_cv.alpha_
```

Out[35]: np.float64(0.49770235643321115)

In [37]:
```
mean_squared_error(y_test, ridge_pred)
```

Out[37]: 0.019544354076968272

In [38]:
```
ridge_cv.coef_
```

Out[38]: array([0.2916599 , 0.0894537 , 0.30197724, 0.13122731, 0.14923577])

In [39]:
```
lasso_cv = LassoCV(alphas=alpha_values,cv=5,random_state=0)
lasso_cv.fit(X_train, y_train)
```

Out[39]:
```
LassoCV(alphas=array([1.00000000e-02, 1.14975700e-02, 1.32194115e-02, 1.51991108e-02,
       1.74752840e-02, 2.00923300e-02, 2.31012970e-02, 2.65608778e-02,
       3.05385551e-02, 3.51119173e-02, 4.03701726e-02, 4.64158883e-02,
       5.33669923e-02, 6.13590727e-02, 7.05480231e-02, 8.11130831e-02,
       9.32603347e-02, 1.07226722e-01, 1.23284674e-01, 1.41747416e-01,
       1.62975083e-01, 1.87381742e-0...
       4.03701726e+02, 4.64158883e+02, 5.33669923e+02, 6.13590727e+02,
       7.05480231e+02, 8.11130831e+02, 9.32603347e+02, 1.07226722e+03,
       1.23284674e+03, 1.41747416e+03, 1.62975083e+03, 1.87381742e+03,
```

In [41]:
```
lasso_pred = lasso_cv.predict(X_test)
```

In [42]:
```
lasso_cv.alpha_
```

Out[42]:
```
np.float64(0.01)
```

In [43]:
```
lasso_cv.coef_
```

Out[43]:
```
array([0.        , 0.        , 0.13504156, 0.07345709, 0.07836195])
```

In [45]:
```
mean_squared_error(y_test, lasso_pred)
```

Out[45]:
```
0.01991781625293486
```

Stacking

In [48]:
```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, AdaBoostClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
```

In [55]:
```
data = pd.read_csv('liver_patient.csv')
y = data.liver_disease
data.drop(['Age', 'Gender', 'liver_disease'], axis=1, inplace=True)
```

In [56]:
```
MM = MinMaxScaler()
X1 = MM.fit_transform(data)
X = pd.DataFrame(X1[:, 0:8])
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.10, random_state=0)
```

In [57]:
```
DT = DecisionTreeClassifier()
BC = BaggingClassifier(n_estimators=10, random_state=0)
PC = BaggingClassifier(n_estimators=10, bootstrap=True, random_state=0)
RFC = RandomForestClassifier(n_estimators=10, max_features="sqrt", random_state=0)
ABC = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1), n_estimators=500, random_state=0)

DT.fit(X_train, y_train)
BC.fit(X_train, y_train)
PC.fit(X_train, y_train)
RFC.fit(X_train, y_train)
ABC.fit(X_train, y_train)

pred_DT = DT.predict(X_test)
pred_BC = BC.predict(X_test)
pred_PC = PC.predict(X_test)
pred_RFC = RFC.predict(X_test)
pred_ABC = ABC.predict(X_test)
```

In [58]:
```
print("Decision Tree Accuracy:",accuracy_score(y_test,pred_DT))
print("Bagging Accuracy:",accuracy_score(y_test,pred_BC))
print("Pasting Accuracy:",accuracy_score(y_test,pred_PC))
print("Random Forest Accuracy:",accuracy_score(y_test,pred_RFC))
print("AdaBoost Accuracy:",accuracy_score(y_test,pred_ABC))
```

```
Decision Tree Accuracy: 0.6440677966101694
Bagging Accuracy: 0.7288135593220338
Pasting Accuracy: 0.7288135593220338
Random Forest Accuracy: 0.7457627118644068
AdaBoost Accuracy: 0.7457627118644068
```

In [59]:
```
estimators=[('dt',DT),('bc',BC),('pc',PC),('rfc',RFC),('abc',ABC)]
stk=StackingClassifier(estimators=estimators, final_estimator=LogisticRegression(), passthrough=True)
stk.fit(X_train,y_train)
pred_stk=stk.predict(X_test)
print("Stacking Accuracy:",accuracy_score(y_test,pred_stk))
```

```
Stacking Accuracy: 0.7288135593220338
```

In [ ]: