# Experiment_8_assessment

February 27, 2025

# 1 Data Mining and machine Learning

# 2 Experiment 8

## 2.1 26 February

# 3 Name: Tufan Kundu

# 4 Reg no.: 24MDT0184

# 5 Decision Tree: Gradient boosting

# 6 Q1. Today we will try to see how gradient boosting can be implemented both manually and using the inbuilt classes.

#### 6.0.1 importing th necessary libraries

[1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

#### 6.0.2 Loading the dataset

[2]:
```
df = pd.read_csv(r"D:\study material\VIT_Data_Science\Winter_Sem\Data Mining␣
↪and Machine Learning Lab\Class_notes\ML_exp2\Book1.csv")
df
```

[2]:

| | price | area | bedrooms | bathrooms | stories | parking | furnishingstatus |
|---|---|---|---|---|---|---|---|
| 0 | 13300000 | 7420 | 4 | 2 | 3 | 2 | furnished |
| 1 | 12250000 | 8960 | 4 | 4 | 4 | 3 | furnished |
| 2 | 12250000 | 9960 | 3 | 2 | 2 | 2 | semi-furnished |
| 3 | 12215000 | 7500 | 4 | 2 | 2 | 3 | furnished |
| 4 | 11410000 | 7420 | 4 | 1 | 2 | 2 | furnished |
| .. | … | … | … | … | … | … | … |

```
244  4550000  5320         3          1        2        0  semi-furnished
245  4550000  5360         3          1        2        2     unfurnished
246  4550000  3520         3          1        1        0  semi-furnished
247  4550000  8400         4          1        4        3     unfurnished
248  4543000  4100         2          2        1        0  semi-furnished

[249 rows x 7 columns]
```

### 6.0.3 Dropping the unnecessary column

[3]:
```
df.drop('furnishingstatus',axis = 1, inplace = True)
df
```

[3]:
```
        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
..        ...   ...       ...        ...      ...      ...
244   4550000  5320         3          1        2        0
245   4550000  5360         3          1        2        2
246   4550000  3520         3          1        1        0
247   4550000  8400         4          1        4        3
248   4543000  4100         2          2        1        0

[249 rows x 6 columns]
```

### 6.0.4 Performing min-max scaling

[4]:
```
scaler = MinMaxScaler()
X = scaler.fit_transform(df)
```

### 6.0.5 Setting the x and y(target) variable

[5]:
```
x = X[:,1:]
y = X[:,0]
```

### 6.0.6 Train test split

[6]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪2,random_state=0)
```

### 6.0.7 Fitting the data in decision tree

[7]:
```
model_dtr = DecisionTreeRegressor()
model_dtr.fit(x_train,y_train)
y_pred_dtr = model_dtr.predict(x_test)
print(f"MSE for decision tree regressor: {mean_squared_error(y_test,y_pred_dtr)}")
print(f"R2 score for decision tree regressor: {r2_score(y_test,y_pred_dtr)}")
```

```
MSE for decision tree regressor: 0.03538319712823755
R2 score for decision tree regressor: -0.43838112172408383
```

## 6.1 Now we will see how we can implement the gradient boosting technique with inbuilt class.

[8]:
```
from sklearn.ensemble import GradientBoostingRegressor
model_gbr = GradientBoostingRegressor(n_estimators = 100, learning_rate = 0.01, max_depth = 3, random_state=0)
model_gbr.fit(x_train,y_train)
y_pred_gbr = model_gbr.predict(x_test)
print(f"MSE for GradientBoosting Regressor: {mean_squared_error(y_test,y_pred_gbr)}")
print(f"R2 score for GradientBoosting Regressor: {r2_score(y_test,y_pred_gbr)}")
```

```
MSE for GradientBoosting Regressor: 0.017811289322634986
R2 score for GradientBoosting Regressor: 0.27594382660242045
```

### 6.1.1 Error reduced significantly by using gradient boosting technique

## 6.2 Perform hyperparameter tuning using GridsearchCV by giving an parameter grid with the hyperparameters n estimators, learning rate and max depth. Each parameter should be having ateast 10 values in the parameter grid.

[9]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators' : [25,50,100,200,250,300,400,450,500,700],
    'learning_rate' : [0.1,0.5,0.01,0.05,0.08,0.001,0.005,0.008,0.0001,0.0005],
    'max_depth' : [1,2,3,4,5,6,7,8,9,10],
}
gbr_model = GradientBoostingRegressor()
grid_search = GridSearchCV(estimator=gbr_model,param_grid=param_grid,cv = 5, scoring='neg_mean_squared_error',n_jobs=-1)
grid_search.fit(x_train,y_train)
```

[9]:
```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(), n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.5, 0.01, 0.05, 0.08, 0.001,
```

```
                                    0.005, 0.008, 0.0001, 0.0005],
                        'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                        'n_estimators': [25, 50, 100, 200, 250, 300, 400, 450,
                                   500, 700]},
             scoring='neg_mean_squared_error')
```

[10]:
```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)

print(f"Best parameters:\n{best_params}")
print(f"Best model :\n{best_model}")
print("\nMSE(optimised after hyperparameter tuning):
 ↪",mean_squared_error(y_test,y_pred))
print("r2 score:",r2_score(y_test,y_pred))
```

```
Best parameters:
{'learning_rate': 0.1, 'max_depth': 1, 'n_estimators': 700}
Best model :
GradientBoostingRegressor(max_depth=1, n_estimators=700)

MSE(optimised after hyperparameter tuning): 0.018494705889616964
r2 score: 0.24816189709905545
```

## 6.3 Similarly you can import GradientBoostingClassifier from sklearn.ensemble. Use the liver patient dataset and fit a Decision tree and GradientBoostingClassfier

### 6.3.1 importing the libraries

[11]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score
```

### 6.3.2 Loading the dataset

[12]:
```
df = pd.read_csv(r"D:\study material\VIT_Data_Science\Winter_Sem\Data Mining␣
 ↪and Machine Learning Lab\Class_notes\ML_exp4\liver_patient.csv")
df
```

[12]:
```
     Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0     65  Female              0.7               0.1                   187
1     62    Male             10.9               5.5                   699
2     62    Male              7.3               4.1                   490
3     58    Male              1.0               0.4                   182
4     72    Male              3.9               2.0                   195
..   …     …               …                 …                    …
578   60    Male              0.5               0.1                   500
```

```
579   40   Male           0.6           0.1                98
580   52   Male           0.8           0.2               245
581   31   Male           1.3           0.5               184
582   38   Male           1.0           0.3               216

     Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                          16                          18             6.8
1                          64                         100             7.5
2                          60                          68             7.0
3                          14                          20             6.8
4                          27                          59             7.3
..                        …                         …             …
578                        20                          34             5.9
579                        35                          31             6.0
580                        48                          49             6.4
581                        29                          32             6.8
582                        21                          24             7.3

     Albumin  Albumin_and_Globulin_Ratio  liver_disease
0        3.3                        0.90              1
1        3.2                        0.74              1
2        3.3                        0.89              1
3        3.4                        1.00              1
4        2.4                        0.40              1
..       …                        …            …
578      1.6                        0.37              0
579      3.2                        1.10              1
580      3.2                        1.00              1
581      3.4                        1.00              1
582      4.4                        1.50              0

[583 rows x 11 columns]
```

### 6.3.3 Dropping unnecessary columns

[13]:
```python
df.drop(['Age','Gender'],axis =1 , inplace=True)
df
```

[13]:
```
     Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0                0.7               0.1                   187
1               10.9               5.5                   699
2                7.3               4.1                   490
3                1.0               0.4                   182
4                3.9               2.0                   195
..               …               …                   …
578              0.5               0.1                   500
579              0.6               0.1                    98
```

```
580              0.8              0.2                    245
581              1.3              0.5                    184
582              1.0              0.3                    216

     Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                          16                          18             6.8
1                          64                         100             7.5
2                          60                          68             7.0
3                          14                          20             6.8
4                          27                          59             7.3
..                        …                          …             …
578                        20                          34             5.9
579                        35                          31             6.0
580                        48                          49             6.4
581                        29                          32             6.8
582                        21                          24             7.3

     Albumin  Albumin_and_Globulin_Ratio  liver_disease
0        3.3                        0.90              1
1        3.2                        0.74              1
2        3.3                        0.89              1
3        3.4                        1.00              1
4        2.4                        0.40              1
..       …                        …            …
578      1.6                        0.37              0
579      3.2                        1.10              1
580      3.2                        1.00              1
581      3.4                        1.00              1
582      4.4                        1.50              0

[583 rows x 9 columns]
```

### 6.3.4 Min max scaling

[14]:
```
X = scaler.fit_transform(df)
x = X[:,:-1]
y = X[:,-1]
```

### 6.3.5 Train test split

[15]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,␣
 ↪random_state=0)
```

[16]:
```
##Fitting in decision tree classifier
model_dtc = DecisionTreeClassifier()
model_dtc.fit(x_train,y_train)
y_pred_dtc = model_dtc.predict(x_test)
```

```
print(f"Accuracy score when fitting the model via decision tree classifier:
↪{accuracy_score(y_test,y_pred_dtc)*100} %")

## Fitting using Gradient boosting classifier
model_gbc = GradientBoostingClassifier(n_estimators=100,learning_rate=0.
↪01,max_depth=3,random_state=0)
model_gbc.fit(x_train,y_train)
y_pred_gbc = model_gbc.predict(x_test)
print(f"Accuracy score when fitting the model via gradient boosting classifier:
↪{accuracy_score(y_test,y_pred_gbc)*100} %")
```

```
Accuracy score when fitting the model via decision tree
classifier:59.82905982905983 %
Accuracy score when fitting the model via gradient boosting
classifier:65.8119658119658 %
```

### 6.4 Q2. Perform hyperparameter tuning on with the hyperparameters n estimators, learning rate and max depth. Each parameter should be having ateast 10 values in the parameter grid. Find the best combination of the parameters to get the better accuracy.

[17]:
```
param_grid = {
    'n_estimators' : [25,50,100,200,250,300,400,450,500,700],
    'learning_rate' : [0.1,0.5,0.01,0.05,0.08,0.001,0.005,0.008,0.0001,0.0005],
    'max_depth' : [1,2,3,4,5,6,7,8,9,10],
}
gbc_model = GradientBoostingClassifier()
grid_search = GridSearchCV(estimator=gbc_model, param_grid=param_grid,␣
↪cv=5,scoring='accuracy', n_jobs=-1)
grid_search.fit(x_train, y_train)
```

[17]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.1, 0.5, 0.01, 0.05, 0.08, 0.001,
                                           0.005, 0.008, 0.0001, 0.0005],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [25, 50, 100, 200, 250, 300, 400, 450,
                                          500, 700]},
             scoring='accuracy')
```

[18]:
```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
y_pred = best_model.predict(x_test)
print(f"Best parameters:\n{best_params}")
print(f"Best model:\n{best_model}")
print(f"\nAccuracy score after hyperparameter tuning:
↪{accuracy_score(y_test,y_pred)*100} %")
```

```
Best parameters:
```

```
{'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 400}
Best model:
GradientBoostingClassifier(learning_rate=0.01, max_depth=1, n_estimators=400)

Accuracy score after hyperparameter tuning:65.8119658119658 %
```

### 6.5 Reguarization techniques: Ridge and lasso regression.

Now we will try to look at ridge and lasso regression which are again regularization tech niques 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

### 6.6 Q3. To perform ridge and lasso regression download the Book1.csv dataset to do house price prediction.

#### 6.6.1 Reading the dataset

[37]:
```
df = pd.read_csv(r"D:\study material\VIT_Data_Science\Winter_Sem\Data Mining␣
↪and Machine Learning Lab\Class_notes\ML_exp2\Book1.csv")
## Dropping unnecessary column
df.drop('furnishingstatus',axis = 1, inplace = True)
df
```

[37]:
```
       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
..       ...   ...       ...        ...      ...      ...
244  4550000  5320         3          1        2        0
245  4550000  5360         3          1        2        2
246  4550000  3520         3          1        1        0
247  4550000  8400         4          1        4        3
248  4543000  4100         2          2        1        0

[249 rows x 6 columns]
```

#### 6.6.2 Min max scaling

[38]:
```
X = scaler.fit_transform(df)
```

### 6.6.3 Setting x and y

[39]:
```
x = X[:,1:]
y = X[:,0]
```

### 6.6.4 Train test split

[40]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪2,random_state=0)
```

[41]:
```
from sklearn.linear_model import RidgeCV, LassoCV

## Defining alpha values for tuning
alpha_values = np.logspace(-2,4,100)
print()
#------------------------------------------------------------------
# Ridge Regression with RidgeCV
#------------------------------------------------------------------
ridge_cv = RidgeCV(alphas = alpha_values, store_cv_values = True)
ridge_cv.fit(x_train,y_train)
ridge_pred = ridge_cv.predict(x_test)
print("Best Ridge Alpha:",ridge_cv.alpha_)
print("Ridge Regression MSE:", mean_squared_error(y_test,ridge_pred))
print("Ridge coefficients:",ridge_cv.coef_)

print()
print()

#------------------------------------------------------------------
#Lasso regression with Lassocv
#------------------------------------------------------------------
lasso_cv = LassoCV(alphas=alpha_values,cv = 5, random_state = 0)
lasso_cv.fit(x_train,y_train)
lasso_pred = lasso_cv.predict(x_test)
print("Best Lasso Alpha:",lasso_cv.alpha_)
print("Lasso Regression MSE:", mean_squared_error(y_test,lasso_pred))
print("lasso coefficients:",lasso_cv.coef_)
```

```
Best Ridge Alpha: 0.49770235643321115
Ridge Regression MSE: 0.019544354076968272
Ridge coefficients: [0.2916599  0.0894537  0.30197724 0.13122731 0.14923577]


Best Lasso Alpha: 0.01
Lasso Regression MSE: 0.01991781625293486
lasso coefficients: [0.         0.         0.13504156 0.07345709 0.07836195]
```

### 6.7 Stacking

[33]:
```
import warnings
warnings.filterwarnings("ignore")
```

[35]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,␣
↪AdaBoostClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression

# Load dataset
data = pd.read_csv(r"D:\study material\VIT_Data_Science\Winter_Sem\Data Mining␣
↪and Machine Learning Lab\Class_notes\ML_exp4\liver_patient.csv")

# Extract target variable (Y) and drop unnecessary columns
Y = data.liver_disease
data.drop(['Age', 'Gender', 'liver_disease'], axis=1, inplace=True)

# Normalize data using MinMaxScaler
scaler = preprocessing.MinMaxScaler()
X_scaled = scaler.fit_transform(data)
X = pd.DataFrame(X_scaled[:, 0:8])  # Retaining first 8 normalized features

# Split dataset into training and testing sets (90% train, 10% test)
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.10,␣
↪random_state=0)

# Define base classifiers
DT = DecisionTreeClassifier()
BC = BaggingClassifier(n_estimators=10, random_state=0)  # Bagging Classifier
PC = BaggingClassifier(n_estimators=10, bootstrap=True, random_state=0)  #␣
↪Pasting Classifier
RFC = RandomForestClassifier(n_estimators=10, max_features="sqrt",␣
↪random_state=0)  # Random Forest
ABC = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=1),␣
↪n_estimators=500, random_state=0)  # AdaBoost

# Train the base models
DT.fit(X_train, y_train)
BC.fit(X_train, y_train)
```

```python
PC.fit(X_train, y_train)
RFC.fit(X_train, y_train)
ABC.fit(X_train, y_train)

# Make predictions
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)

# Print accuracy of individual models
print(f"Decision Tree Accuracy:{accuracy_score(y_test, pred_DT)*100} %")
print(f"Bagging Accuracy:{accuracy_score(y_test, pred_BC)*100} %")
print(f"Pasting Accuracy:{accuracy_score(y_test, pred_PC)*100} %")
print(f"Random Forest Accuracy:{accuracy_score(y_test, pred_RFC)*100} %")
print(f"AdaBoost Accuracy:{accuracy_score(y_test, pred_ABC)*100} %")

# Define Stacking Classifier with Logistic Regression as the final estimator
estimators = [('dt', DT), ('bc', BC), ('pc', PC), ('rfc', RFC), ('abc', ABC)]
stk = StackingClassifier(estimators=estimators,␣
↪final_estimator=LogisticRegression(), passthrough=True)

# Train the stacking classifier
stk.fit(X_train, y_train)

# Make predictions with stacking classifier
pred_stk = stk.predict(X_test)

# Print accuracy of Stacking Classifier
print(f"Stacking Accuracy:{accuracy_score(y_test, pred_stk)*100} %")
```

```
Decision Tree Accuracy:66.10169491525424 %
Bagging Accuracy:72.88135593220339 %
Pasting Accuracy:72.88135593220339 %
Random Forest Accuracy:74.57627118644068 %
AdaBoost Accuracy:77.96610169491525 %
Stacking Accuracy:72.88135593220339 %
```