2. MODEL TRAINING

December 28, 2024

0.1 Model Training

1.1 Import Data and Required Packages

Importing Pandas, Numpy, Matplotlib, Seaborn and Warings Library.

[39]:
```
# Basic Import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Modelling
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor,AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression, Ridge,Lasso
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import RandomizedSearchCV
#from catboost import CatBoostRegressor
from xgboost import XGBRegressor
import warnings
```

Import the CSV Data as Pandas DataFrame

[40]:
```
df = pd.read_csv('data/stud.csv')
```

Show Top 5 Records

[41]:
```
df.head()
```

[41]:
```
   gender race_ethnicity parental_level_of_education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B             master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test_preparation_course  math_score  reading_score  writing_score
0                    none          72             72             74
```

```
1             completed          69          90          88
2                  none          90          95          93
3                  none          47          57          44
4                  none          76          78          75
```

Preparing X and Y variables

[42]:
```python
X = df.drop(columns=['math_score'],axis=1)
```

[43]:
```python
X.head()
```

[43]:
```
   gender race_ethnicity parental_level_of_education         lunch  \
0  female        group B           bachelor's degree      standard
1  female        group C                some college      standard
2  female        group B             master's degree      standard
3    male        group A          associate's degree  free/reduced
4    male        group C                some college      standard

  test_preparation_course  reading_score  writing_score
0                    none             72             74
1               completed             90             88
2                    none             95             93
3                    none             57             44
4                    none             78             75
```

[44]:
```python
print("Categories in 'gender' variable:     ",end=" " )
print(df['gender'].unique())

print("Categories in 'race_ethnicity' variable:  ",end=" ")
print(df['race_ethnicity'].unique())

print("Categories in'parental level of education' variable:",end=" " )
print(df['parental_level_of_education'].unique())

print("Categories in 'lunch' variable:     ",end=" " )
print(df['lunch'].unique())

print("Categories in 'test preparation course' variable:     ",end=" " )
print(df['test_preparation_course'].unique())
```

```
Categories in 'gender' variable:      ['female' 'male']
Categories in 'race_ethnicity' variable:   ['group B' 'group C' 'group A' 'group
D' 'group E']
Categories in'parental level of education' variable: ["bachelor's degree" 'some
college' "master's degree" "associate's degree"
 'high school' 'some high school']
Categories in 'lunch' variable:      ['standard' 'free/reduced']
Categories in 'test preparation course' variable:      ['none' 'completed']
```

[45]:
```
y = df['math_score']
```

[46]:
```
# Create Column Transformer with 3 types of transformers
num_features = X.select_dtypes(exclude="object").columns
cat_features = X.select_dtypes(include="object").columns

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer

numeric_transformer = StandardScaler()
oh_transformer = OneHotEncoder()

preprocessor = ColumnTransformer(
    [
        ("OneHotEncoder", oh_transformer, cat_features),
         ("StandardScaler", numeric_transformer, num_features),
    ]
)
```

[47]:
```
X = preprocessor.fit_transform(X)
```

[48]:
```
X.shape
```

[48]: (1000, 19)

[52]:
```
# separate dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪2,random_state=42)
X_train.shape, X_test.shape
```

[52]: ((800, 19), (200, 19))

Create an Evaluate Function to give all metrics after model Training

[53]:
```
def evaluate_model(true, predicted):
    mae = mean_absolute_error(true, predicted)
    mse = mean_squared_error(true, predicted)
    rmse = np.sqrt(mean_squared_error(true, predicted))
    r2_square = r2_score(true, predicted)
    return mae, rmse, r2_square
```

[54]:
```
models = {
    "Linear Regression": LinearRegression(),
    "Lasso": Lasso(),
    "Ridge": Ridge(),
    "K-Neighbors Regressor": KNeighborsRegressor(),
    "Decision Tree": DecisionTreeRegressor(),
```

```
    "Random Forest Regressor": RandomForestRegressor(),
    "XGBRegressor": XGBRegressor(),
    ##"CatBoosting Regressor": CatBoostRegressor(verbose=False),
    "AdaBoost Regressor": AdaBoostRegressor()
}
model_list = []
r2_list =[]

for i in range(len(list(models))):
    model = list(models.values())[i]
    model.fit(X_train, y_train) # Train model

    # Make predictions
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    # Evaluate Train and Test dataset
    model_train_mae , model_train_rmse, model_train_r2 =␣
 ↪evaluate_model(y_train, y_train_pred)

    model_test_mae , model_test_rmse, model_test_r2 = evaluate_model(y_test,␣
 ↪y_test_pred)


    print(list(models.keys())[i])
    model_list.append(list(models.keys())[i])

    print('Model performance for Training set')
    print("- Root Mean Squared Error: {:.4f}".format(model_train_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_train_mae))
    print("- R2 Score: {:.4f}".format(model_train_r2))

    print('----------------------------------')

    print('Model performance for Test set')
    print("- Root Mean Squared Error: {:.4f}".format(model_test_rmse))
    print("- Mean Absolute Error: {:.4f}".format(model_test_mae))
    print("- R2 Score: {:.4f}".format(model_test_r2))
    r2_list.append(model_test_r2)

    print('='*35)
    print('\n')
```

```
Linear Regression
Model performance for Training set
- Root Mean Squared Error: 5.3231
- Mean Absolute Error: 4.2667
```

```
- R2 Score: 0.8743
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3940
- Mean Absolute Error: 4.2148
- R2 Score: 0.8804
===================================


Lasso
Model performance for Training set
- Root Mean Squared Error: 6.5938
- Mean Absolute Error: 5.2063
- R2 Score: 0.8071
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.5197
- Mean Absolute Error: 5.1579
- R2 Score: 0.8253
===================================


Ridge
Model performance for Training set
- Root Mean Squared Error: 5.3233
- Mean Absolute Error: 4.2650
- R2 Score: 0.8743
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.3904
- Mean Absolute Error: 4.2111
- R2 Score: 0.8806
===================================


K-Neighbors Regressor
Model performance for Training set
- Root Mean Squared Error: 5.7077
- Mean Absolute Error: 4.5167
- R2 Score: 0.8555
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 7.2530
- Mean Absolute Error: 5.6210
- R2 Score: 0.7838
===================================
```

```
Decision Tree
Model performance for Training set
- Root Mean Squared Error: 0.2795
- Mean Absolute Error: 0.0187
- R2 Score: 0.9997
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 8.3039
- Mean Absolute Error: 6.5950
- R2 Score: 0.7166
===================================


Random Forest Regressor
Model performance for Training set
- Root Mean Squared Error: 2.2926
- Mean Absolute Error: 1.8174
- R2 Score: 0.9767
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 5.9606
- Mean Absolute Error: 4.5912
- R2 Score: 0.8540
===================================


XGBRegressor
Model performance for Training set
- Root Mean Squared Error: 1.0073
- Mean Absolute Error: 0.6875
- R2 Score: 0.9955
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.4733
- Mean Absolute Error: 5.0577
- R2 Score: 0.8278
===================================


AdaBoost Regressor
Model performance for Training set
- Root Mean Squared Error: 5.8285
- Mean Absolute Error: 4.7908
- R2 Score: 0.8493
----------------------------------
Model performance for Test set
- Root Mean Squared Error: 6.0334
- Mean Absolute Error: 4.6801
```

```
- R2 Score: 0.8504
===================================
```

0.1.1 Results

[55]:
```
pd.DataFrame(list(zip(model_list, r2_list)), columns=['Model Name',␣
↪'R2_Score']).sort_values(by=["R2_Score"],ascending=False)
```

[55]:

	Model Name	R2_Score
2	Ridge	0.880593
0	Linear Regression	0.880433
5	Random Forest Regressor	0.853994
7	AdaBoost Regressor	0.850404
6	XGBRegressor	0.827797
1	Lasso	0.825320
3	K-Neighbors Regressor	0.783813
4	Decision Tree	0.716629

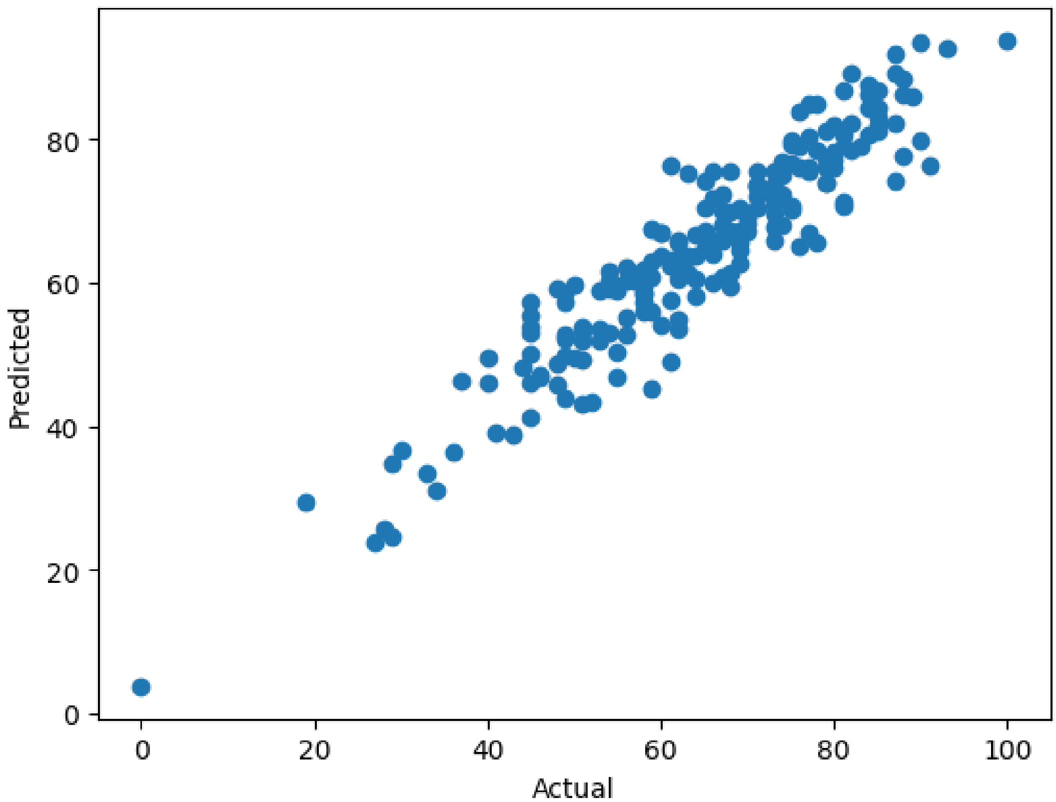
0.2 Linear Regression

[56]:
```
lin_model = LinearRegression(fit_intercept=True)
lin_model = lin_model.fit(X_train, y_train)
y_pred = lin_model.predict(X_test)
score = r2_score(y_test, y_pred)*100
print(" Accuracy of the model is %.2f" %score)
```

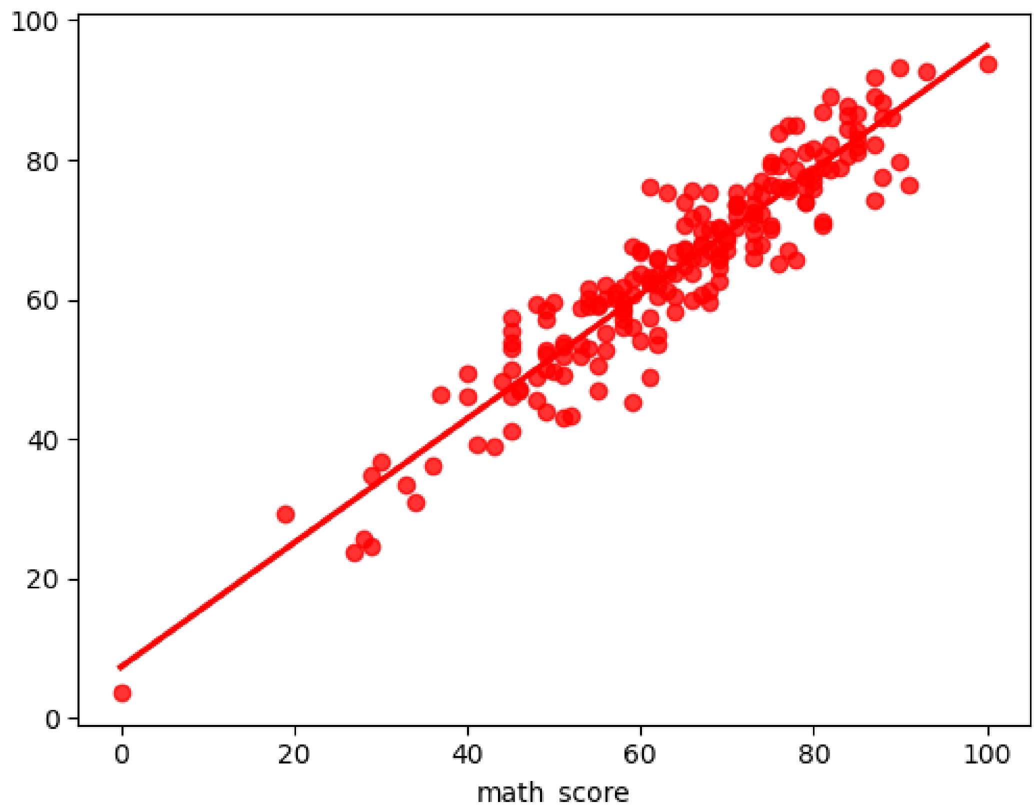
```
 Accuracy of the model is 88.04
```

0.3 Plot y_pred and y_test

[23]:
```
plt.scatter(y_test,y_pred);
plt.xlabel('Actual');
plt.ylabel('Predicted');
```



[24]:
```
sns.regplot(x=y_test,y=y_pred,ci=None,color ='red');
```



Difference between Actual and Predicted Values

[25]:
```
pred_df=pd.DataFrame({'Actual Value':y_test,'Predicted Value':
    ↪y_pred,'Difference':y_test-y_pred})
pred_df
```

[25]:

	Actual Value	Predicted Value	Difference
521	91	76.387970	14.612030
737	53	58.885970	-5.885970
740	80	76.990265	3.009735
660	74	76.851804	-2.851804
411	84	87.627378	-3.627378
..	…	…	…
408	52	43.409149	8.590851
332	62	62.152214	-0.152214
208	74	67.888395	6.111605
613	65	67.022287	-2.022287
78	61	62.345132	-1.345132

[200 rows x 3 columns]