

Model Optimization and Tuning Phase

***GARMENT
WORKER
PRODUCTIVITY
PREDICTION***

Team ID: 1720673861

Date: 05/07/2024

TEAM:

SHREYAS REDDY GUVVALA

NIKHIL PULAGAM

MULE VIGNESH

1. Model Optimization and Tuning Phase

5.1 Hyperparameter Tuning Documentation

To check the model performance on test and train data, we calculate the RMSE score. The RMSE score tells us how far off the predictions of the model are, on average, from the actual values.

Activity 1.1: Linear Regression Model

```
# training score
mse = mean_squared_error(y_train, predict_train)
rmse_lr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_train)
```

Root Mean Squared Error: 0.16226529653729893

```
# testing score
mse = mean_squared_error(y_test, predict_test)
rmse_lr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_lr_test)
```

Root Mean Squared Error: 0.16116562949494187

Activity 1.2: Decision Tree Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_dtr)
rmse_dtr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_dtr_train)
```

Root Mean Squared Error: 0.13187559206436333

```
# testing score
mse = mean_squared_error(y_test, predict_test_dtr)
rmse_dtr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_dtr_test)
```

Root Mean Squared Error: 0.12918875831022705

Activity 1.3: Random Forest Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_rfr)
rmse_rfr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_rfr_train)
```

Root Mean Squared Error: 0.13066329578222882

```
# testing score
mse = mean_squared_error(y_test, predict_test_rfr)
rmse_rfr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_rfr_test)
```

Root Mean Squared Error: 0.12721255996349562

Activity 1.4:

Gradient

Boosting Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_gbr)
rmse_gbr_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_train)
```

Root Mean Squared Error: 0.14244277376076936

```
# testing score
mse = mean_squared_error(y_test, predict_test_gbr)
rmse_gbr_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_gbr_test)
```

Root Mean Squared Error: 0.1394815884261522

Activity 1.5:

Extreme

Gradient Boost Regressor Model

```
# training score
mse = mean_squared_error(y_train, predict_train_xgb)
rmse_xgb_train = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_xgb_train)
```

Root Mean Squared Error: 0.057574476336590054

```
# testing score
mse = mean_squared_error(y_test, predict_test_xgb)
rmse_xgb_test = np.sqrt(mse)
print('Root Mean Squared Error:', rmse_xgb_test)
```

Root Mean Squared Error: 0.12343275775750703

Activity 1.6: Bagging Regressor Model

```
# Evaluate performance
train_rmse_b = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse_b = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("Bagging Regressor:")
print(f"Training RMSE: {train_rmse_b}")
print(f"Testing RMSE: {test_rmse_b}")
```

```
Bagging Regressor:
Training RMSE: 0.11003378916688866
Testing RMSE: 0.11605208654607595
```

Activity 1.7: Boosting Regressor Model

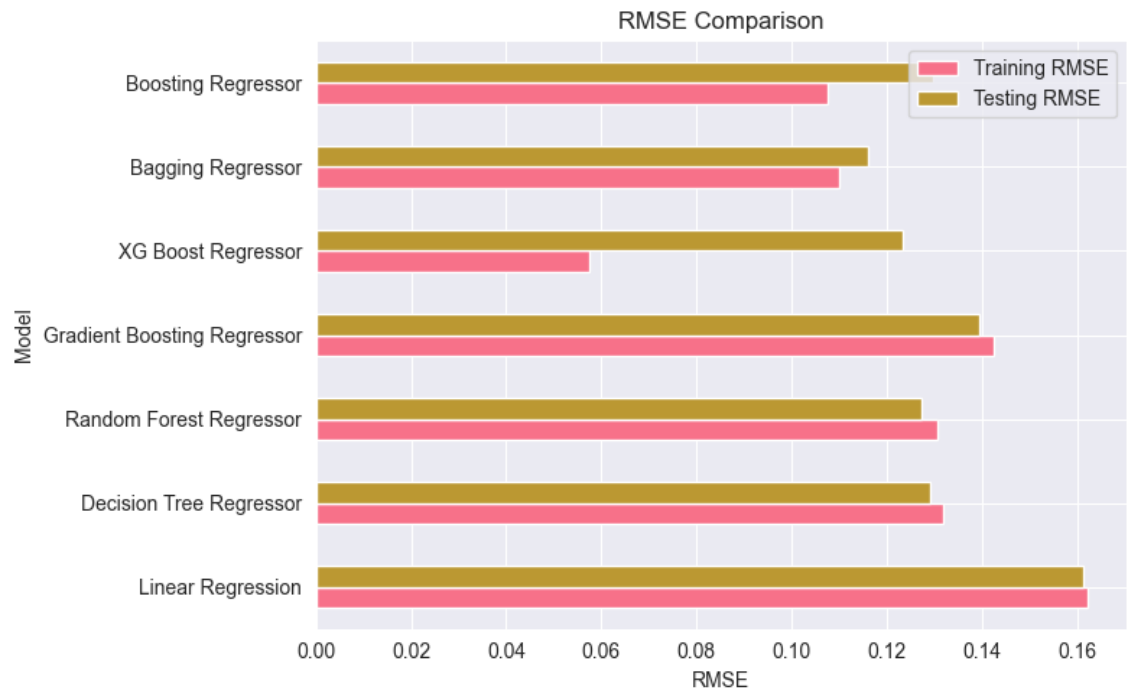
```
# Evaluate performance
train_rmse_bo = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse_bo = np.sqrt(mean_squared_error(y_test, y_test_pred))

print("AdaBoost Regressor:")
print(f"Training RMSE: {train_rmse_bo}")
print(f"Testing RMSE: {test_rmse_bo}")
|
```

```
AdaBoost Regressor:
Training RMSE: 0.1075964215336082
Testing RMSE: 0.1297762247652773
```

5.2 Performance Metrics Comparison Report

This code creates a Pandas Data Frame named "results" that contains the model names and root mean squared errors (RMSE) for both the training and testing data for each of the seven regression models: Linear Regression, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, XG Boost Regressor, Bagging Regressor, and Boosting Regressor.



To determine which model is best, we should look for the model with the lowest RMSE value on the test data. This suggests that the model predicts value closer to the actual values. In this out of the seven models selected Boosting Regressor satisfies the conditions and hence selected.

5.3 Final Model Selection Justification

```
# dumping the selected model
pickle.dump(boosting_reg, open('productivity.pkl', 'wb'))
```

This code uses the "pickle" library in Python to save the trained Boosting Regressor model named "boosting_reg" as a file named "productivity.pkl". The "dump" method from the pickle library is used to save the model object in a serialized form that can be used again later. The "wb" parameter indicates that the file should be opened in binary mode to write data to it.

In this section, we will be building a web application that would help us integrate the machine learning model we have built and trained.

A user interface is provided for the users to enter the values for predictions. The entered values are fed into the saved model, and the prediction is displayed on the UI.

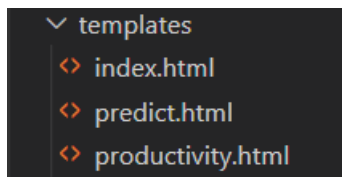
The section has following task:

- Building HTML pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project we create three html files:

- index.html
 - predict.html
 - productivity.html
- and save these html files in the templates folder.



Activity 2.2: Build Python code:

Importing the libraries

```
import pickle
from flask import Flask, render_template, request
import pandas as pd
import numpy as np
```

This code first loads the saved Boosting Regressor model from the "productivity.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
model1 = pickle.load(open('productivity.pkl', 'rb'))
app=Flask(__name__)
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render_template()" method is used to render an HTML template

named "index.html". The "index.html" is the home page.

```
@app.route('/')  
def home():  
    return render_template('index.html')
```

The route in this case is "/predict". When a user accesses the "/predict" route of the website, this function "index()" is called.

The "render_template()" method is used to render an HTML template named "predict.html".

```
@app.route('/predict') # rendering the html template  
def index():  
    return render_template('predict.html')
```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/data_predict", and the method is set to GET and POST. The function "predict()" is then associated with this route.

The input data is collected from an HTML form and includes information such as the quarter, department, day of the week, team number, time allocated, unfinished items, overtime, incentive, idle time, idle men, style change, and number of workers.

The code converts some of the input data into a format that can be used by the machine learning model. For example, the department input is converted from a string to a binary value (1 for sewing, 0 for finishing), and the day of the week input is converted to a numerical value (0-6).

The model is then used to make a prediction based on the input data. The prediction is rounded to 4 decimal places and multiplied by 100 to convert it to a percentage.

The prediction value is passed to the HTML template 'productivity.html' where it is displayed as a text message. The message informs the user of the predicted productivity based on the input data.

```
@app.route('/data_predict', methods=['GET','POST'])
def predict() :
```

```
    quarter = int(request.form['Quarter'])
```

```
    department = request.form['Department']
```

```
    if department == 'Sewing':
```

```
        department = 1
```

```
    if department == 'sewing':
```

```
        department = 1
```

```
    if department == 'Finishing':
```

```
        department = 0
```

```
    if department == 'finishing':
```

```
        department = 0
```

```
    day = request.form['Day of the week']
```

```
    if day == 'Monday':
```

```
        day = 0
```

```
    if day == 'Tuesday':
```

```
        day = 4
```

```
    if day == 'Wednesday':
```

```
        day = 5
```

```
    if day == 'Thursday':
```

```
        day = 3
```

```
    if day == 'Saturday':
```

```
        day = 1
```

```
    if day == 'Sunday':
```

```
        day = 2
```

```
team = int(request.form['Team Number'])
```

```
time = int(request.form['Time Allocated'])
```

```
items = int(request.form['Unfinished Items'])
```

```
over_time = int(request.form['Over time'])
```

```
incentive = int(request.form['Incentive'])
```

```
idle_time = int(request.form['Idle Time'])
```

```
idle_men = int(request.form['Idle Men'])
```

```
style = int(request.form['Style Change'])
```

```
workers = int(request.form['Number of Workers'])
```

```
prediction = model1.predict(pd.DataFrame([[quarter,department,day,team,time,items,over_time,incentive,idle_time,idle_men,style,workers]], columns=['quarter', 'department', 'day', 'team', 'time', 'items', 'over_time', 'incentive', 'idle_time', 'idle_men', 'style_change', 'no_of_workers']))
```

```
prediction = (np.round(prediction,4))*100
```

```
return render_template('productivity.html', prediction_text ="is {}".format(prediction))
```

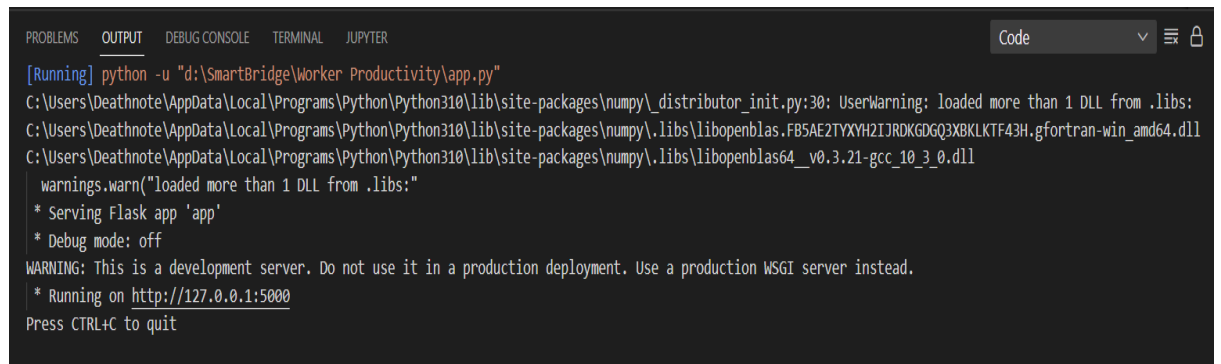
Main Function:

This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask development server.

```
if __name__ == '__main__':  
    app.run()
```

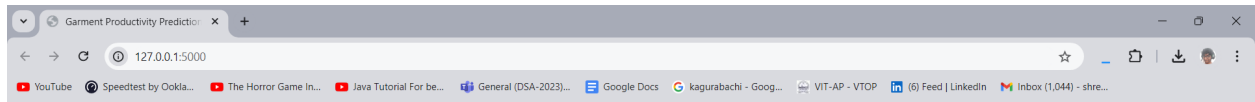
Activity 2.3: Run the web application

When you run the "main.py" file this window will open in the output terminal. Copy the <http://127.0.0.1:5000> and paste this link in your browser.



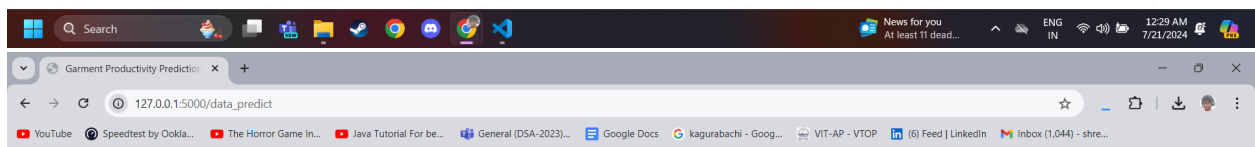
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER  
[Running] python -u "d:\SmartBridge\Worker Productivity\app.py"  
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:  
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas.FB5AE2TVXVH2IJRDKGDGQ3XBKLTf43H.gfortran-win_amd64.dll  
C:\Users\Deathnote\AppData\Local\Programs\Python\Python310\lib\site-packages\numpy\.libs\libopenblas64_v0.3.21-gcc_10_3_0.dll  
  warnings.warn("loaded more than 1 DLL from .libs:")  
* Serving Flask app 'app'  
* Debug mode: off  
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit
```

IMAGES OF HTML SITE:



Welcome to Garment Productivity Prediction

Please navigate to the [Prediction Page](#) to make predictions.

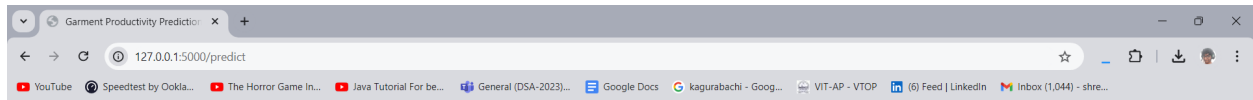


Predicted Productivity

The predicted productivity is: The predicted productivity is: [82.35]%

[Make Another Prediction](#)





Predict Garment Productivity

Quarter (1-4):

Department (Sewing or Finishing):

Day of the week:

Team Number:

Time Allocated:

Unfinished Items:

Over Time:

Incentive:

Idle Time:

Idle Men:

Style Change:

Number of Workers:

