```
# -*- coding: utf-8 -*-
"""sih.ipynb

Automatically generated by
Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1WtsrIZVvRvjQVJTiv5_sHLqFtQNlteLv
"""

i
mport pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from
sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import
mean_squared_error, r2_score

# Load your dataset (you can replace the file path with your
dataset)
df = pd.read_csv('/content/drive/MyDrive/Datasets/ai4i2020.csv')

# Encode categorical
variables using Label Encoding
label_encoder = LabelEncoder()
df['Type'] =
label_encoder.fit_transform(df['Type'])

# Feature Engineering
df['temp_diff'] = df['Process
temperature [K]'] - df['Air temperature [K]']
df['overstrain'] = df['Tool wear [min]'] *
df['Torque [Nm]']

# Create the target variable 'machine failure'
df['machine failure'] = (

(df['Tool wear [min]'].between(200, 240)) |
    ((df['Air temperature [K]'] - df['Process
temperature [K]'] < 8.6) & (df['Rotational speed [rpm]'] < 1380)) |
    ((df['Torque
[Nm]'] * (df['Rotational speed [rpm]'] * 2 * 3.14159265359 / 60) < 3500)) |
    ((df['Torque
[Nm]'] * df['Tool wear [min]'] > 11000) & (df['Type'] == 'L')) |
    ((df['Torque [Nm]']
* df['Tool wear [min]'] > 12000) & (df['Type'] == 'M')) |
    ((df['Torque [Nm]'] *
df['Tool wear [min]'] > 13000) & (df['Type'] == 'H')) |
    (df['Type'] ==
'R')
).astype(int)

# One-hot encode categorical columns
categorical_columns = ['Product
ID']
df_encoded = pd.get_dummies(df, columns=categorical_columns,
drop_first=True)

df_encoded.head()

# Drop unnecessary columns
df = df.drop(columns=['Product
ID', 'UDI', 'Product ID', 'Machine failure', 'Type'])

# Normalize numerical features using
Min-Max scaling
scaler = MinMaxScaler()
df[['Air temperature [K]', 'Process temperature [K]',
'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]', 'temp_diff', 'overstrain']] =
scaler.fit_transform(df[['Air temperature [K]', 'Process temperature [K]', 'Rotational speed
[rpm]', 'Torque [Nm]', 'Tool wear [min]', 'temp_diff', 'overstrain']])

df.head()
```

```
# Initialize
the Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(n_estimators=100,
max_depth=3, learning_rate=0.1, random_state=42)

# Split the data into features (X) and target
(y)
X = df.drop(columns=['machine failure','TWF','HDF','PWF','OSF','RNF'])
y = df['machine
failure']

# Train the model on the entire dataset
gb_regressor.fit(X, y)

# Take input values
from the user
print("Enter the values for prediction:")
air_temp =
float(input("Air temperature [K]: "))
process_temp = float(input("Process
temperature [K]: "))
rotational_speed = float(input("Rotational speed [rpm]:
"))
torque = float(input("Torque [Nm]: "))
tool_wear = float(input("Tool
wear [min]: "))

# Calculate additional features
temp_diff = process_temp -
air_temp
overstrain = tool_wear * torque

# Create a DataFrame for prediction
new_data =
pd.DataFrame({
    'Air temperature [K]': [air_temp],
    'Process temperature [K]':
[process_temp],
    'Rotational speed [rpm]': [rotational_speed],
    'Torque [Nm]': [torque],

   'Tool wear [min]': [tool_wear],
    'temp_diff': [temp_diff],
    'overstrain':
[overstrain]
})

# Predict the failure using the trained model
predicted_failure =
gb_regressor.predict(new_data)

# Display the prediction result
if predicted_failure[0] == 1:

  print("The machine is predicted to fail.")
else:
    print("The machine is
predicted to continue operating.")

from sklearn.metrics import confusion_matrix
import
seaborn as sns
import matplotlib.pyplot as plt
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)
# Predict the machine failures on the test data
y_pred =
gb_regressor.predict(X_test)

# Round the predicted probabilities to binary values (0 or
1)
```

```
y_pred_binary = (y_pred > 0.5).astype(int)

# Calculate the confusion matrix
conf_matrix
= confusion_matrix(y_test, y_pred_binary)

# Create a heatmap to visualize the confusion
matrix
plt.figure(figsize=(6, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
cbar=False, square=True,
            xticklabels=['Predicted Negative', 'Predicted Positive'],

           yticklabels=['Actual Negative', 'Actual Positive'])
plt.xlabel('Predicted
Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

from
sklearn.model_selection import train_test_split
from sklearn.linear_model import
LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import
RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import
mean_squared_error, r2_score, f1_score, accuracy_score



# Split the data into training (80%)
and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize different regression models
models = {
    'Linear Regression':
LinearRegression(),
    'Decision Tree Regression': DecisionTreeRegressor(random_state=42),

'Random Forest Regression': RandomForestRegressor(random_state=42),
    'Gradient Boosting
Regression': GradientBoostingRegressor(random_state=42)
}

# Initialize dictionaries to store
evaluation results
mse_results = {}
r2_results = {}
f1_results = {}
accuracy_results = {}

#
Train and evaluate each model
for model_name, model in models.items():
    # Train the model on
the training data
    model.fit(X_train, y_train)

    # Make predictions on the testing data

  y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse =
mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    f1 = f1_score(y_test,
(y_pred > 0.5).astype(int))
    accuracy = accuracy_score(y_test, (y_pred >
0.5).astype(int))
```

```
    # Store the results in the respective dictionaries

mse_results[model_name] = mse
    r2_results[model_name] = r2
    f1_results[model_name] = f1

  accuracy_results[model_name] = accuracy

# Display the evaluation results for all
models
print("Mean Squared Error (MSE):")
for model_name, mse in
mse_results.items():
    print(f"{model_name}: {mse:.2f}")

print("\nR-squared
(R2):")
for model_name, r2 in r2_results.items():
    print(f"{model_name}:
{r2:.2f}")

print("\nF1 Score:")
for model_name, f1 in f1_results.items():

print(f"{model_name}: {f1:.2f}")

print("\nAccuracy:")
for model_name,
accuracy in accuracy_results.items():
    print(f"{model_name}:
{accuracy:.2f}")

from sklearn.metrics import accuracy_score

# Threshold the predicted
probabilities to binary values (0 or 1)
y_pred_binary = (y_pred > 0.5).astype(int)

#
Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred_binary)

# Display
the accuracy
print(f"Accuracy: {accuracy * 100:.2f}%")

from sklearn.metrics import
fbeta_score

# Predict the machine failures on the test data
y_pred =
gb_regressor.predict(X_test)

# Set a list of beta values to calculate F-beta
scores
beta_values = [0.5, 1, 2]  # You can adjust these values as needed

# Calculate and
display F-beta scores for each beta value
for beta in beta_values:
    fbeta =
fbeta_score(y_test, (y_pred > 0.5).astype(int), beta=beta)
    print(f"F-{beta} Score:
{fbeta:.2f}")

from sklearn.metrics import f1_score

# Predict the machine failures on the
test data
y_pred = gb_regressor.predict(X_test)

# Round the predicted probabilities to binary
values (0 or 1)
y_pred_binary = (y_pred > 0.5).astype(int)
```

```
# Calculate the F1 score
f1 =
f1_score(y_test, y_pred_binary)

# Display the F1 score
print(f"F1 Score:
{f1:.2f}")

from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Predict the machine failures on the test data
y_pred =
gb_regressor.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse =
mean_squared_error(y_test, y_pred)

# Calculate Root Mean Squared Error (RMSE)
rmse =
mean_squared_error(y_test, y_pred, squared=False)

# Calculate Mean Absolute Error (MAE)
mae =
mean_absolute_error(y_test, y_pred)

# Calculate R-squared (R2)
r2 = r2_score(y_test,
y_pred)

# Display the regression metrics
print(f"Mean Squared Error (MSE):
{mse:.2f}")
print(f"Root Mean Squared Error (RMSE):
{rmse:.2f}")
print(f"Mean Absolute Error (MAE):
{mae:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```