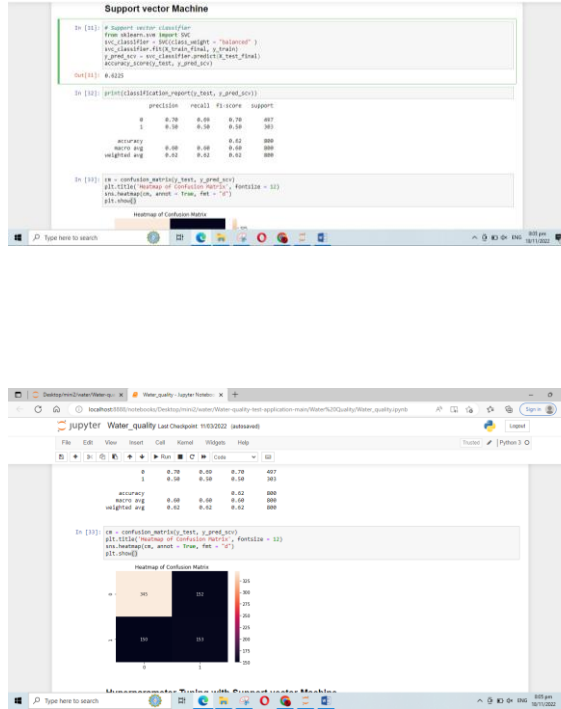


Project Development Phase Model Performance Test

Date	10 November 2022
Team ID	PNT2022TMID13654
Project Name	Efficient water quality analysis and prediction using machine learning
Maximum Marks	10 Marks

Model Performance Testing:

S.N o.	Parameter	Values	Screenshot
1.	Metrics	<p>Classification Model: # Support vector classifier from sklearn.svm import SVC svc_classifier = SVC(class_weight = "balanced") svc_classifier.fit(X_train_final, y_train) y_pred_scv = svc_classifier.predict(X_test_final) ACCURACY: accuracy_score(y_test, y_pred_scv) #0.62255 CLASSIFICATION REPORT : print(classification_report(y_test, y_pred_scv)) CONFUSION MATRIX: cm = confusion_matrix(y_test, y_pred_scv) plt.title('Heatmap of Confusion Matrix', fontsize = 12) sns.heatmap(cm, annot = True, fmt = "d") plt.show()</p>	

```
# Random Forest Classifier
from sklearn.ensemble import
RandomForestClassifier
rf_classifier =
RandomForestClassifier(n_estimators
= 20, criterion = 'entropy',
class_weight =
"balanced_subsample",random_state
= 51)
rf_classifier.fit(X_train_final, y_train)
y_pred =
rf_classifier.predict(X_test_final)
ACCURACY_SCORE:
accuracy_score(y_test, y_pred)
CLASSIFICATION REPORT:
print(classification_report(y_test,
y_pred))
# XGBoost Classifier
from xgboost import XGBClassifier
xgb_classifier =
XGBClassifier(random_state=0)
xgb_classifier.fit(X_train_final, y_train)
y_pred_xgb =
xgb_classifier.predict(X_test_final)
ACCURACY_SCORE:
accuracy_score(y_test, y_pred_xgb)
CLASSIFICATION REPORT:
print(classification_report(y_test,
y_pred_xgb))
```

```
In [27]: # Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', class_weight = 'balanced_subsample', random_state = 51)
rf_classifier.fit(X_train_final, y_train)
y_pred = rf_classifier.predict(X_test_final)
accuracy_score(y_test, y_pred)

Out[27]: 0.82875

In [28]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.65	0.87	0.74	407
1	0.52	0.34	0.43	303
accuracy	0.58	0.55	0.54	800
weighted avg	0.58	0.55	0.54	800

```
In [29]: # XGBoost Classifier
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier(random_state=0)
xgb_classifier.fit(X_train_final, y_train)
y_pred_xgb = xgb_classifier.predict(X_test_final)
accuracy_score(y_test, y_pred_xgb)

Out[29]: 0.83225
```

```
In [28]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.65	0.87	0.74	407
1	0.52	0.34	0.43	303
accuracy	0.58	0.55	0.54	800
weighted avg	0.58	0.55	0.54	800

```
In [29]: # XGBoost Classifier
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier(random_state=0)
xgb_classifier.fit(X_train_final, y_train)
y_pred_xgb = xgb_classifier.predict(X_test_final)
accuracy_score(y_test, y_pred_xgb)

Out[29]: 0.83225

In [30]: print(classification_report(y_test, y_pred_xgb))
```

	precision	recall	f1-score	support
0	0.67	0.77	0.72	407
1	0.50	0.30	0.40	303
accuracy	0.58	0.57	0.57	800
weighted avg	0.57	0.57	0.57	800

2. Tune the Model

Hyperparameter Tuning

```
param_grid = {'C': [0.1, 1, 10, 100, 200
, 400 , 600 , 800],
'gamma': [1, 0.1, 0.01, 0.001,
0.0001],
'kernel': ['rbf']}
```

```
from sklearn.model_selection import
GridSearchCV
grid = GridSearchCV(SVC(),
param_grid, refit = True, verbose = 3)
```

```
# fitting the model for grid search
grid.fit(X_train_final, y_train)
# print best parameter after tuning
print(grid.best_params_)
```

```
# print how our model looks after
hyper-parameter tuning
print(grid.best_estimator_)
```

The screenshot displays a Jupyter Notebook interface with the following content:

```

In [10]: # defining parameter space
param_grid = {'C': [0.1, 1, 10, 100, 1000, 10000, 100000],
              'gamma': [0.001, 0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000],
              'kernel': ['rbf']}

In [11]: from sklearn.model_selection import GridSearchCV

In [12]: grid = GridSearchCV(svm, param_grid, refit = True, verbose = 5)

# fitting the model for grid search
grid.fit(X_train,X_test,X_val)

# Printing 5 folds for each of 40 candidates, totaling 200 Folds
(CV 0/1) .....-0.1, gamma=1, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=10, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=100, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=1000, kernel=rbf, total time= 0.40
(CV 0/1) .....-0.1, gamma=10000, kernel=rbf, total time= 0.40
(CV 0/1) .....-0.1, gamma=100000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=1, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=10, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=100, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=1000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=10000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=100000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=1, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=10, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=100, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=1000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=10000, kernel=rbf, total time= 0.30
(CV 0/1) .....-0.1, gamma=100000, kernel=rbf, total time= 0.30

In [13]: # print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)

(CV 0/1) .....gamma=0.001, kernel='rbf'

(CV 0/1) .....gamma=0.001, kernel='rbf'
  
```