


Project Development Phase Model Performance Test

Date	10 November 2022
Team ID	PNT2022TMID52728
Project Name	Project – Efficient Water Quality Analysis and Prediction Using Machine Learning
Maximum Marks	10 Marks

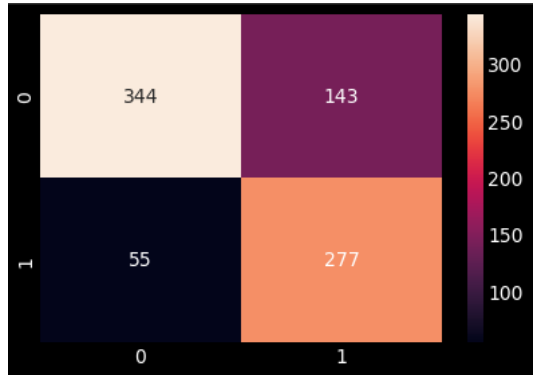
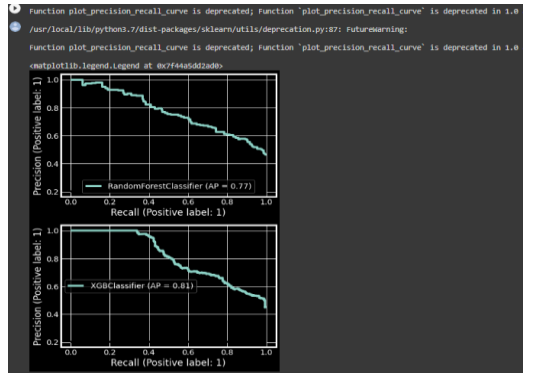
Model Performance Testing:

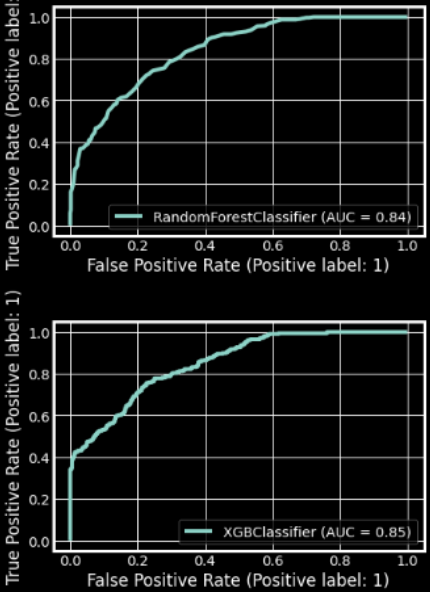
Project team shall fill the following information in model performance testing template.

S.No.	Parameter	Values	Screenshot
1.	Regression Model	<pre>from sklearn.ensemble import RandomForestRegressor regressor = RandomForestRegressor(n_estimators = 10, random_state = 0) regressor.fit(x_train, y_train) y_pred = regressor.predict(x_test) from sklearn import metrics print('MAE:',metrics.mean_absolute_err or(y_test,y_pred)) print('MSE:',metrics.mean_squared_erro r(y_test,y_pred)) print('RMSE:',np.sqrt(metrics.mean_squ ared_error(y_test,y_pred))) MAE: 1.013774436090232 MSE: 6.2406858345864675 RMSE: 2.498136472370248 #accuracy of the model metrics.r2_score(y_test, y_pred) 0.9659820315121997</pre>	 <p>The screenshot displays the output of the provided Python code. It shows the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and the coefficient of determination (R2 score) for the regression model. The values are: MAE: 1.013774436090232, MSE: 6.2406858345864675, RMSE: 2.498136472370248, and R2 score: 0.9659820315121997.</p>

2.	Hyperparameter tuning	<pre> SPACE = [skopt.space.Real(0.01, 0.5, name='learning_rate', prior='log-uniform'), skopt.space.Integer(1, 30, name='max_depth'), skopt.space.Integer(2, 100, name='num_leaves'), skopt.space.Real(0.1, 1.0, name='feature_fraction', prior='uniform'), skopt.space.Real(0.1, 1.0, name='subsample', prior='uniform')] @skopt.utils.use_named_args(SPACE) def objective(**params): return -1.0 * train_evaluate(params) results = skopt.forest_minimize(objective, SPACE, n_calls=30, n_random_starts=10) best_auc = -1.0 * results.fun best_params = results.x print('best result: ', best_auc) print('best parameters: ', best_params) </pre>	<pre> best result: 0.6509559162948146 best parameters: [0.014589467657194726, 21, 26, 0.972340211773363, 0.6065207062490089] </pre>
3.	Validation Method	<pre> def train_evaluate(search_params): path = "water_potability.csv" data = pd.read_csv(path) X = data.drop(['Sulfate', 'Potability'], axis=1) y = data['Potability'] X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=1234) train_data = lgb.Dataset(X_train, label=y_train) valid_data = lgb.Dataset(X_valid, label=y_valid, reference=train_data) params = {'objective': 'binary', 'metric': 'auc', **search_params} model = lgb.train(params, train_data, num_boost_round=300, early_stopping_rounds=30, valid_sets=[valid_data], valid_names=['valid']) score = model.best_score['valid']['auc'] return score if __name__ == '__main__': score = train_evaluate(SEARCH_PARAMS) print('validation AUC:', score) </pre>	<pre> validation AUC: 0.6509559162948146 </pre>

4.	Model Summary	<pre>{'entity': {'hybrid_pipeline_software_specs': [], 'software_spec': {'id': 'acd9c798-6974-5d2f-a657-ce06e986df4d', 'name': 'tensorflow_rt22.1-py3.9'}, 'type': 'tensorflow_2.7'}, 'metadata': {'created_at': '2022-11-15T15:37:29.698Z', 'id': 'd36fff39-3076-4cdb-986d-ce425e593a5e', 'modified_at': '2022-11-15T15:37:34.096Z', 'name': 'Water Quality Analysis', 'owner': 'IBMid-6630040G8W', 'resource_key': '6793bd5d-1bb9-472e-8fad-e72d691cc411', 'space_id': 'f33d596f-6c60-4ff8-b7d9-51d79c0fd8ce'}, 'system': {'warnings': []}}}</pre>	<pre>In [65]: model_details = client.repository.store_model(model='waterquality.tgz',meta_props={ client.repository.ModelMetadata.NAME:'Water Quality Analysis', client.repository.ModelMetadata.TYPE:'tensorflow_2.7', client.repository.ModelMetadata.SOFTWARE_SPEC_UID:software_space_uid }) In [66]: model_details Out[66]: {'entity': {'hybrid_pipeline_software_specs': [], 'software_spec': {'id': 'acd9c798-6974-5d2f-a657-ce06e986df4d', 'name': 'tensorflow_rt22.1-py3.9'}, 'type': 'tensorflow_2.7'}, 'metadata': {'created_at': '2022-11-15T15:37:29.698Z', 'id': 'd36fff39-3076-4cdb-986d-ce425e593a5e', 'modified_at': '2022-11-15T15:37:34.096Z', 'name': 'Water Quality Analysis', 'owner': 'IBMid-6630040G8W', 'resource_key': '6793bd5d-1bb9-472e-8fad-e72d691cc411', 'space_id': 'f33d596f-6c60-4ff8-b7d9-51d79c0fd8ce'}, 'system': {'warnings': []}}}</pre>																												
5.	Accuracy	<pre>ModelTrain_Accuracy Test_Accuracy 0 LogisticRegression() 0.490742 0.511600 1 DecisionTreeClassifier() 0.733616 0.697192 2 GaussianNB() 0.538703 0.581197 3 (DecisionTreeClassifier(max_fea tures='auto', r... 0.790210 0.772894 4 LinearSVC() 0.491073 0.511600 5 XGBClassifier() 0.761762 0.758242</pre>	<pre>[] print('The comparison') modelScore = pd.DataFrame({'Model': model, 'Train_Accuracy': trainAccuracy, 'Test_Accuracy': testAccuracy}) modelScore</pre> <table><thead><tr><th></th><th>Model</th><th>Train_Accuracy</th><th>Test_Accuracy</th></tr></thead><tbody><tr><td>0</td><td>LogisticRegression()</td><td>0.490742</td><td>0.511600</td></tr><tr><td>1</td><td>DecisionTreeClassifier()</td><td>0.733616</td><td>0.697192</td></tr><tr><td>2</td><td>GaussianNB()</td><td>0.538703</td><td>0.581197</td></tr><tr><td>3</td><td>(DecisionTreeClassifier(max_features='auto', r...</td><td>0.790210</td><td>0.772894</td></tr><tr><td>4</td><td>LinearSVC()</td><td>0.491073</td><td>0.511600</td></tr><tr><td>5</td><td>XGBClassifier()</td><td>0.761762</td><td>0.758242</td></tr></tbody></table>		Model	Train_Accuracy	Test_Accuracy	0	LogisticRegression()	0.490742	0.511600	1	DecisionTreeClassifier()	0.733616	0.697192	2	GaussianNB()	0.538703	0.581197	3	(DecisionTreeClassifier(max_features='auto', r...	0.790210	0.772894	4	LinearSVC()	0.491073	0.511600	5	XGBClassifier()	0.761762	0.758242
	Model	Train_Accuracy	Test_Accuracy																												
0	LogisticRegression()	0.490742	0.511600																												
1	DecisionTreeClassifier()	0.733616	0.697192																												
2	GaussianNB()	0.538703	0.581197																												
3	(DecisionTreeClassifier(max_features='auto', r...	0.790210	0.772894																												
4	LinearSVC()	0.491073	0.511600																												
5	XGBClassifier()	0.761762	0.758242																												
6.	Confusion Matrix (Random Forest Classifier)	<pre>print('Random Forest Classifier\n') Rfc = RandomForestClassifier() Rfc.fit(X_train, y_train) y_Rfc = Rfc.predict(X_test) print(metrics.classification_report(y_t est, y_Rfc)) print(modelAccuracy.append(metrics. accuracy_score(y_test, y_Rfc))) sns.heatmap(confusion_matrix(y_test, y_Rfc), annot=True, fmt='d') plt.show()</pre>	<table><thead><tr><th></th><th>Actual 0</th><th>Actual 1</th></tr></thead><tbody><tr><th>Predicted 0</th><td>379</td><td>108</td></tr><tr><th>Predicted 1</th><td>71</td><td>261</td></tr></tbody></table>		Actual 0	Actual 1	Predicted 0	379	108	Predicted 1	71	261																			
	Actual 0	Actual 1																													
Predicted 0	379	108																													
Predicted 1	71	261																													

	Confusion Matrix (XGB Classifier)	<pre>print('XGB Classifier\n') xgb = XGBClassifier() xgb.fit(X_train, y_train) y_xgb = xgb.predict(X_test) print(metrics.classification_report(y_test, y_xgb)) print(modelAccuracy.append(metrics.accuracy_score(y_test, y_xgb))) sns.heatmap(confusion_matrix(y_test, y_xgb), annot=True, fmt='d') plt.show()</pre>																															
7.	Precision Recall F1 Score (Random Forest Classifier)	<pre>print('Random Forest Classifier\n') Rfc = RandomForestClassifier() Rfc.fit(X_train, y_train) y_Rfc = Rfc.predict(X_test) print(metrics.classification_report(y_test, y_Rfc)) print(modelAccuracy.append(metrics.accuracy_score(y_test, y_Rfc)))</pre>	<pre>Random Forest Classifier</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.84</td><td>0.78</td><td>0.81</td><td>487</td></tr><tr><td>1</td><td>0.71</td><td>0.79</td><td>0.74</td><td>332</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.78</td><td>819</td></tr><tr><td>macro avg</td><td>0.77</td><td>0.78</td><td>0.78</td><td>819</td></tr><tr><td>weighted avg</td><td>0.79</td><td>0.78</td><td>0.78</td><td>819</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.84	0.78	0.81	487	1	0.71	0.79	0.74	332	accuracy			0.78	819	macro avg	0.77	0.78	0.78	819	weighted avg	0.79	0.78	0.78	819
	precision	recall	f1-score	support																													
0	0.84	0.78	0.81	487																													
1	0.71	0.79	0.74	332																													
accuracy			0.78	819																													
macro avg	0.77	0.78	0.78	819																													
weighted avg	0.79	0.78	0.78	819																													
	Precision Recall F1 Score (XGB Classifier)	<pre>print('XGB Classifier\n') xgb = XGBClassifier() xgb.fit(X_train, y_train) y_xgb = xgb.predict(X_test) print(metrics.classification_report(y_test, y_xgb)) print(modelAccuracy.append(metrics.accuracy_score(y_test, y_xgb)))</pre>	<pre>XGB Classifier</pre> <table><thead><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr></thead><tbody><tr><td>0</td><td>0.86</td><td>0.71</td><td>0.78</td><td>487</td></tr><tr><td>1</td><td>0.66</td><td>0.83</td><td>0.74</td><td>332</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.76</td><td>819</td></tr><tr><td>macro avg</td><td>0.76</td><td>0.77</td><td>0.76</td><td>819</td></tr><tr><td>weighted avg</td><td>0.78</td><td>0.76</td><td>0.76</td><td>819</td></tr></tbody></table>		precision	recall	f1-score	support	0	0.86	0.71	0.78	487	1	0.66	0.83	0.74	332	accuracy			0.76	819	macro avg	0.76	0.77	0.76	819	weighted avg	0.78	0.76	0.76	819
	precision	recall	f1-score	support																													
0	0.86	0.71	0.78	487																													
1	0.66	0.83	0.74	332																													
accuracy			0.76	819																													
macro avg	0.76	0.77	0.76	819																													
weighted avg	0.78	0.76	0.76	819																													
8.	Precision-Recall or PR curve	<pre>from scikitplot.metrics import plot_roc_curve from sklearn.metrics import plot_precision_recall_curve plot_precision_recall_curve(Rfc,X_test,y_test) plt.plot([0,1], [0.2035,0.2035], c='k') plt.legend(loc='best') plot_precision_recall_curve(xgb,X_test,y_test) plt.plot([0,1], [0.2035,0.2035], c='k') plt.legend(loc='best')</pre>																															

9.	PR vs ROC curve	<pre>plot_roc_curve(Rfc,X_test,y_test) plot_roc_curve(xgb,X_test,y_test)</pre>	<div><div>Function plot_roc_curve is deprecated; Function :func:'plot_roc_curve'</div><div><sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f44a5ffa550></div></div> 
----	-----------------	--------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------