

Project Development Phase V Model Performance Test

Date	18 November 2022
Team ID	591761
Project Name	Predicting Mental Health Illness Of Working Professionals Using Machine Learning
Maximum Marks	10 Marks

Model Performance Testing:

S.No.	Parameter	Values	Screenshot																													
1		<div>Classification Model: Confusion Matrix - True Positive: 45.87% False Positive: 4.53% True Negative: 41.07% False Positive: 8.53%</div>	<div><pre># Generating the confusion matrix for the tuned AdaBoostClassifier cf_matrix = confusion_matrix(y_test, pred_abc_tuned) # Creating a heatmap for the confusion matrix plt.figure(figsize=(8, 6)) sns.heatmap(cf_matrix / np.sum(cf_matrix), annot=True, fmt='.2%', plt.title('Confusion Matrix of AdaBoost Classifier after tuning') plt.xlabel('Predicted') plt.ylabel('Actual') plt.show()</pre></div> <div><table><caption>Confusion Matrix of AdaBoost Classifier after tuning</caption><thead><tr><th></th><th>0</th><th>1</th></tr></thead><tbody><tr><th>0</th><td>41.07%</td><td>8.53%</td></tr><tr><th>1</th><td>4.53%</td><td>45.87%</td></tr></tbody></table></div>		0	1	0	41.07%	8.53%	1	4.53%	45.87%																				
		0	1																													
	0	41.07%	8.53%																													
1	4.53%	45.87%																														
		<div>Accuray Score - 0.86933</div>	<div><pre># Creating a tuned AdaBoostClassifier with the best parameters abc_tuned = AdaBoostClassifier(random_state=49, n_estimators=11, learning_rate=1.02) # Fitting the model to the training data abc_tuned.fit(X_train, y_train) # Making predictions on the test set pred_abc_tuned = abc_tuned.predict(X_test) # Calculating and printing the accuracy of the tuned AdaBoostClassifier from sklearn.metrics import accuracy_score print('Accuracy of AdaBoost (tuned) =', accuracy_score(y_test, pred_abc_tuned))</pre></div> <div>Accuracy of AdaBoost (tuned) = 0.8693333333333333</div>																													
Metrics	Classification Report - 0.87	<div><pre># Classification report for the tuned AdaBoost classifier print("\nClassification Report for Tuned AdaBoost Classifier:") print(classification_report(y_test, pred_abc_tuned))</pre></div> <div><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>0.90</td><td>0.83</td><td>0.86</td><td>186</td></tr><tr><td>1</td><td>0.84</td><td>0.91</td><td>0.88</td><td>189</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>375</td></tr><tr><td>macro avg</td><td>0.87</td><td>0.87</td><td>0.87</td><td>375</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.87</td><td>0.87</td><td>375</td></tr></tbody></table></div>		precision	recall	f1-score	support	0	0.90	0.83	0.86	186	1	0.84	0.91	0.88	189	accuracy			0.87	375	macro avg	0.87	0.87	0.87	375	weighted avg	0.87	0.87	0.87	375
	precision	recall	f1-score	support																												
0	0.90	0.83	0.86	186																												
1	0.84	0.91	0.88	189																												
accuracy			0.87	375																												
macro avg	0.87	0.87	0.87	375																												
weighted avg	0.87	0.87	0.87	375																												

2		<p>Hyperparameter Tuning - 0.8693</p> <p>(We can see an improvement of 0.5% from non boosted model)</p>	<pre> import numpy as np from sklearn.model_selection import RandomizedSearchCV params_abc = { 'n_estimators': [int(x) for x in np.linspace(start=1, stop=50, num=15)], 'learning_rate': [0.07 + x / 100 for x in range(0, 43)] } # Assuming 'abc' is your Gradient Boosting Classifier # Replace 'estimator=abc' with your actual Gradient Boosting Classifier instance abc_random = RandomizedSearchCV(estimator=abc, param_distributions=params_abc, n_iter=50, cv=5, n_jobs=-1) params_abc {'n_estimators': [1, 4, 8, 11, 15, 18, 22, 25, 29, 32, 36, 39, 43, 46, 50], 'learning_rate': [0.07, 0.08, 0.09, 1.0, 1.01, 1.02, 1.03, 1.04]} abc_random.fit(X_train,y_train) RandomizedSearchCV(cv=5, estimator=AdaBoostClassifier(random_state=99), n_iter=50, n_jobs=-1, param_distributions={'learning_rate': [0.07, 0.08, 0.09, 1.0, 1.01, 1.02, 1.03, 1.04], 'n_estimators': [1, 4, 8, 11, 15, 18, 22, 25, 29, 32, 36, 39, 43, 46, 50]}, random_state=99) abc_random.best_params_ {'n_estimators': 11, 'learning_rate': 1.02} # Creating a tuned AdaBoostClassifier with the best parameters abc_tuned = AdaBoostClassifier(random_state=40, n_estimators=11, learning_rate=1.02) # Fitting the model to the training data abc_tuned.fit(X_train, y_train) # Making predictions on the test set pred_abc_tuned = abc_tuned.predict(X_test) # Calculating and printing the accuracy of the tuned AdaBoostClassifier from sklearn.metrics import accuracy_score print('Accuracy of AdaBoost (tuned) : ', accuracy_score(y_test, pred_abc_tuned)) Accuracy of AdaBoost (tuned) = 0.8693333333333333 </pre>
	Tune the Model	<p>Validation Method - Train-Test Split:</p> <p>Score for Logistic regression is: 0.848</p> <p>Score for KNN Classifier is: 0.776</p> <p>Score for Decision Tree Classifier is: 0.7946666666666666</p> <p>Score for Random Forest Classifier is: 0.8533333333333334</p> <p>Score for AdaBoost Classifier is: 0.864</p> <p>Score for Gradient Boosting Classifier is: 0.84</p> <p>Score for XGB Classifier is: 0.84</p>	<pre> from sklearn.metrics import accuracy_score def model_test(X_train, X_test, y_train, y_test, model, model_name): model.fit(X_train, y_train) y_pred = model.predict(X_test) accuracy = accuracy_score(y_test, y_pred) print(f'Score for {model_name} is: {accuracy}') print() # Loop through the models in the model_dict and test each one for model_name, model in model_dict.items(): model_test(X_train, X_test, y_train, y_test, model, model_name) Score for Logistic regression is: 0.848 Score for KNN Classifier is: 0.776 Score for Decision Tree Classifier is: 0.7946666666666666 C:\Users\mp4vl\anaconda3\lib\site-packages\sklearn\neighbors_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. 'skew', 'kurtosis'), the default behavior of 'mode' typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of 'keepdims' will become False, the 'axis' over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set 'keepdims' to True or False to avoid this warning. mode, _ = stats.mode(_y[neigh_ind, k], axis=1) Score for Random Forest Classifier is: 0.8533333333333334 Score for AdaBoost Classifier is: 0.864 Score for Gradient Boosting Classifier is: 0.84 Score for XGB Classifier is: 0.84 </pre>