Model Optimization and Tuning Phase Template

Date	10 July 2024
Team ID	SWTID1720075414
Project Title	Panic Disorder Detection
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining machine learning models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (6 Marks): (Not mandatory for this project)

<table>
<tr><th>Model</th><th>Tuned Hyperparameters</th><th>Optimal Values</th></tr>
<tr><td>Decision Tree</td><td><pre>
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5,10,15],
    'min_samples_split': [2,3,10],
    'min_samples_leaf': [1,2,3],
    'max_features': [None, 'sqrt', 'log2']
}

#creating a decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=1234)

#Create GridSearchCV object
grid_search = GridSearchCV(dt_classifier, param_grid, cv=5, verbose = 1, n_jobs=-1)

#Fit the data to perform grid search
grid_search.fit(x_res_train[fts], y_res_train)
</pre></td><td><pre>
#Print the best hyperparameteers
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Score", grid_search.best_score_)

Fitting 5 folds for each of 216 candidates, totalling 1080
Best Hyperparameters: {'criterion': 'gini', 'max_depth': N
Best Score 0.9914015567048008
</pre></td></tr>
<tr><td>Random Forest</td><td><pre>
param_grid = {
    'n_estimators': [50,100,200],
    'max_depth': [None, 5,10],
    'min_samples_split': [2,5,10],
    'min_samples_leaf': [1,2,4],
    'max_features': ['sqrt', 'log2']
}

#creating a Ramdom Forest Classifier
rf_classifier = RandomForestClassifier(random_state=1234)

#Create GridSearchCV object
grid_search = GridSearchCV(rf_classifier, param_grid=param_grid, cv=5, verbose = 1, n_jobs=4)

#Fit the data to perform grid search
grid_search.fit(x_res_train[fts], y_res_train)
</pre></td><td><pre>
#Print the best hyperparameteers
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Score", grid_search.best_score_)

Fitting 5 folds for each of 162 candidates, totalling 81
Best Hyperparameters: {'max_depth': None, 'max_features':
Best Score 0.9909784255341378
</pre></td></tr>
</table>

XGBoost	`param_grid = {`
	`    'min_child_weight': [10,20],`
	`    'gamma': [0,1.5,2.0],`
	`    'colsample_bytree': [0.6,0.8,0.9],`
	`    'max_depth': [4,5,6],`
	`}`
	
	`xgb = xgboost.XGBClassifier(learning_rate=0.5, n_estimators=100, objective='binary:logistic', nthread=3)`
	
	`fitmodel = GridSearchCV(xgb, param_grid=param_grid, cv=5, refit=True, scoring = 'accuracy', n_jobs=4, verbose=3)`
	
	`#Fit the data to perform grid search`
	`fitmodel.fit(x_res_train[fts], y_res_train)`
		`#Print the best hyperparameteers`
		`print("Best Hyperparameters:", fitmodel.best_params_)`
		`print("Best Score", fitmodel.best_score_)`
		
		`Fitting 5 folds for each of 54 candidates, totalling 270 fits`
		`Best Hyperparameters: {'colsample_bytree': 0.9, 'gamma': 0, 'm`
		`Best Score 0.9890926187118007`

KNN	`param_grid = {`
	`    'n_neighbors': [3,5,7],`
	`    'weights': ['uniform', 'distance'],`
	`    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],`
	`    'leaf_size': [20,30,40],`
	`    'p': [1,2]`
	`}`
	
	`knn = KNeighborsClassifier()`
	
	`#Create GridSearchCV object`
	`grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, verbose = 1, n_jobs=4)`
	
	`#Fit the data to perform grid search`
	`grid_search.fit(x_res_train[fts], y_res_train)`
		`# Print the best parameters and best score`
		`print("Best Parameters: ", grid_search.best_params_)`
		`print("Best Score: ", grid_search.best_score_)`
		
		`Fitting 5 folds for each of 144 candidates, totalling`
		`Best Parameters:  {'algorithm': 'brute', 'leaf_size':`
		`Best Score:  0.7771295215869312`

Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric

Decision Tree	`y_pred = grid_search.best_estimator_.predict(x_test[fts])`
	`print(confusion_matrix(y_test,y_pred))`
	`print(classification_report(y_test,y_pred))`
	`print("SCORE:",grid_search.best_estimator_.score(x_test[fts],y_test))`
	
	`[[18898   261]`
	` [    4   837]]`
	`              precision    recall  f1-score   support`
	
	`           0       1.00      0.99      0.99     19159`
	`           1       0.76      1.00      0.86       841`
	
	`    accuracy                           0.99     20000`
	`   macro avg       0.88      0.99      0.93     20000`
	`weighted avg       0.99      0.99      0.99     20000`
	
	`SCORE: 0.98675`

<table>
<tr><td>Random Forest</td><td>

```
y_pred = grid_search.best_estimator_.predict(x_test[fts])
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("SCORE:",grid_search.best_estimator_.score(x_test[fts],y_test))
```

```
[[18848   311]
 [   14   827]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     19159
           1       0.73      0.98      0.84       841

    accuracy                           0.98     20000
   macro avg       0.86      0.98      0.91     20000
weighted avg       0.99      0.98      0.98     20000

SCORE: 0.98375
```

</td></tr>
<tr><td>KNN</td><td>

```
y_pred=grid_search.best_estimator_.predict(x_test[fts])
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("SCORE:",grid_search.best_estimator_.score(x_test[fts],y_test))
```

```
[[13375  5784]
 [  565   276]]
              precision    recall  f1-score   support

           0       0.96      0.70      0.81     19159
           1       0.05      0.33      0.08       841

    accuracy                           0.68     20000
   macro avg       0.50      0.51      0.44     20000
weighted avg       0.92      0.68      0.78     20000

SCORE: 0.68255
```

</td></tr>
<tr><td>XGBoost</td><td>

```
y_pred = fitmodel.best_estimator_.predict(x_test[fts])
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print("SCORE:",fitmodel.best_estimator_.score(x_test[fts],y_test))
```

```
[[16546  2613]
 [    7   834]]
              precision    recall  f1-score   support

           0       1.00      0.86      0.93     19159
           1       0.24      0.99      0.39       841

    accuracy                           0.87     20000
   macro avg       0.62      0.93      0.66     20000
weighted avg       0.97      0.87      0.90     20000

SCORE: 0.869
```

</td></tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Decision Tree	Decision Tree Model was chosen because it displayed an incredible amount of accuracy when run through hyperparameter tuning. It was able to reduce overfitting very well, which fits the project perfectly, making it the perfect for selecting it as the final model for this project.