ModelOptimizationandTuningPhaseTemplate

Date	16th may 2025
TeamID	LTVIP2025TMID60515
Project Title	Revolutionizing Liver Care: Predicting Liver CirrhosisUsingAdvancedMachineLearning Techniques.
Maximum Marks	10 Marks

ModelOptimizationandTuningPhase

TheModelOptimizationandTuningPhaseinvolvesrefiningmachinelearningmodelsforpeak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performancemetrics,andjustifyingthefinalmodelselectionforenhancedpredictiveaccuracy and efficiency.

HyperparameterTuningDocumentation(6Marks):

Model	TunedHyperparameters	Optimal Values

Naive Bayes	No hyperparameters to tune for GaussianNB,directlyfittingandscoring	`Train score: 0.8353096179183136`
		`Test score: 0.7789473684210526`
		`Accuracy on test set: 0.7789473684210526`

Random Forest	`rf = RandomForestClassifier()`
	
	`# Hyperparameter grid`
	`param_dist = {`
	`'n_estimators': [100, 200, 300, 400, 500],`
	`'max_depth': [None, 10, 20, 30, 40, 50],`
	`'min_samples_split': [2, 5, 10],`
	`'min_samples_leaf': [1, 2, 4],`
	`'bootstrap': [True, False]`
	`}`
		`print('Best Hyperparameters for Random Forest:', rf_best_params)`
		`print('Train score:', rf_train_score)`
		`print('Test score:', rf_test_score)`
		`Best Hyperparameters for Random Forest: {'n_estimators': 400, 'min_samples_split': 10,`
		`Train score: 0.930171277997365`
		`Test score: 0.8736842105263158`

Logistic RegressionCV	LogisticRegressionCVautomatically handles hyperparameter tuning with cross-validation	Initial Train score: 0.8840579710144928 Initial Test score: 0.8157894736842105

Ridge Classifier	# Hyperparameter grid for tuning
	param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}
	
	# GridSearchCV for hyperparameter tuning
	grid_search_rg = GridSearchCV(rg, param_grid, cv=5, n_jobs=-1)
	grid_search_rg.fit(X_train, y_train)
	
	# Get the best parameters
	rg_best_params = grid_search_rg.best_params_
		Optimal hyperparameters for Ridge Classifier: {'alpha': 100}
		Accuracy on test set: 0.8210526315789474

SupportVector Classifier	# Reduced hyperparameter grid for quicker tuning
	param_grid = {
	'C': [0.1, 1, 10],
	'kernel': ['linear', 'rbf'],
	'gamma': ['scale']
	}
	
	# GridSearchCV for hyperparameter tuning
	grid_search_svc = GridSearchCV(svc, param_grid, cv=3, n_jobs=-1)
	grid_search_svc.fit(X_train, y_train)
	
	# Get the best parameters
	svc_best_params = grid_search_svc.best_params_
		Accuracy on test set: 0.64
		Initial Train score: 0.7127799736495388
		Initial Test score: 0.6421052631578947

Logistic Regression	# Hyperparameter grid for tuning
	param_grid = {'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2', 'elasticnet', 'none']}
	
	# GridSearchCV for hyperparameter tuning
	grid_search_log = GridSearchCV(log, param_grid, cv=5, n_jobs=-1)
	grid_search_log.fit(X_train, y_train)
	
	# Get the best parameters
	log_best_params = grid_search_log.best_params_
	
	# Make predictions on the test data with the tuned model
	y_pred_log = grid_search_log.predict(X_test)
		Optimal hyperparameters for Logistic Regression: {'C': 0.01, 'penalty': 'l2'}
		Accuracy on test set: 0.8052631578947368

XG Boost	# Simplified hyperparameter grid for tuning
	param_dist = {
	'n_estimators': [100, 150],
	'max_depth': [3, 6],
	'learning_rate': [0.01, 0.1],
	'subsample': [0.7, 1.0]
	}
	
	# RandomizedSearchCV for hyperparameter tuning with fewer iterations
	random_search_xgb = RandomizedSearchCV(model, param_dist, n_iter=5, cv=3, n_jobs=-1, verbose=1)
	random_search_xgb.fit(X_train, y_train)
	
	# Get the best parameters
	xgb_best_params = random_search_xgb.best_params_
		Initial Train score: 0.9920948616600791
		Initial Test score: 0.8421052631578947
		Accuracy on test set: 0.84

<table>
<tr>
<td>KNN</td>
<td><pre>
HYPERPARAMETER TUNING

k = np.random.randint(1,50,60)
params = {'n_neighbors' : k}

random_search = RandomizedSearchCV(knn, params, n_iter=5, cv=5, n_jobs=-1, verbose = 0)
random_search.fit(X_train, y_train)

print('train_score - '+ str(random_search.score(X_train, y_train)))
print('test_score- ' + str(random_search.score(X_test,y_test)))

knn.get_params()
</pre></td>
<td><pre>
Train score with tuned model: 0.8089591567852438
Test score with tuned model: 0.7210526315789474
Optimal hyperparameters for KNN: {'n_neighbors': 21}
Accuracy on test set: 0.72
</pre></td>
</tr>
</table>

Performance Metrics Comparison Report (2 Marks):

<table>
<tr>
<th>Model</th>
<th>Optimized Metric</th>
</tr>
<tr>
<td>Naive Bayes</td>
<td><pre>
Confusion Matrix (Naive Bayes):
 [[49 19]
 [23 99]]
Classification Report (Naive Bayes):
              precision   recall f1-score   support

           0   0.68   0.72   0.70   68
           1   0.84   0.81   0.82   122

    accuracy   0.78   190
   macro avg   0.76   0.77   0.76   190
weighted avg   0.78   0.78   0.78   190
</pre></td>
</tr>
<tr>
<td>Random Forest</td>
<td><pre>
Confusion Matrix (Random Forest):
 [[51 17]
 [8 114]]
Classification Report (Random Forest):
              precision   recall f1-score   support

           0   0.86   0.75   0.80   68
           1   0.87   0.93   0.90   122

    accuracy   0.87   190
   macro avg   0.87   0.84   0.85   190
weighted avg   0.87   0.87   0.87   190
</pre></td>
</tr>
<tr>
<td>LogisticRegression
CV</td>
<td><pre>
Confusion Matrix (Logistic Regression CV):
 [[43 25]
 [10 112]]
Classification Report (Logistic Regression CV):
              precision   recall f1-score   support

           0   0.81   0.63   0.71   68
           1   0.82   0.92   0.86   122

    accuracy   0.82   190
   macro avg   0.81   0.78   0.79   190
weighted avg   0.82   0.82   0.81   190
</pre></td>
</tr>
</table>

Ridge Classifier	<pre>Confusion Matrix (Ridge Classifier):
	 [[44 24]
	 [10 112]]
	Classification Report (Ridge Classifier):
	              precision recall f1-score support
	
	           0 0.81 0.65 0.72 68
	           1 0.82 0.92 0.87 122
	
	    accuracy 0.82 190
	   macro avg 0.82 0.78 0.79 190
	weighted avg 0.82 0.82 0.82 190</pre>

SupportVector Classifier	<pre>Confusion Matrix (Support Vector Classifier):
	 [[6 62]
	 [6 116]]
	Classification Report (Support Vector Classifier):
	              precision recall f1-score support
	
	           0 0.50 0.09 0.15 68
	           1 0.65 0.95 0.77 122
	
	    accuracy 0.64 190
	   macro avg 0.58 0.52 0.46 190
	weighted avg 0.60 0.64 0.55 190</pre>

Logistic Regression	<pre>Confusion Matrix (Logistic Regression):
	 [[42 26]
	 [11 111]]
	Classification Report (Logistic Regression):
	              precision recall f1-score support
	
	           0 0.79 0.62 0.69 68
	           1 0.81 0.91 0.86 122
	
	    accuracy 0.81 190
	   macro avg 0.80 0.76 0.78 190
	weighted avg 0.80 0.81 0.80 190</pre>

XG Boost	<pre>Confusion Matrix (XGBoost):
	 [[48 20]
	 [10 112]]
	Classification Report (XGBoost):
	              precision recall f1-score support
	
	           0 0.83 0.71 0.76 68
	           1 0.85 0.92 0.88 122
	
	    accuracy 0.84 190
	   macro avg 0.84 0.81 0.82 190
	weighted avg 0.84 0.84 0.84 190</pre>

KNN	Confusion Matrix (KNN): [[40 28] [25 97]] Classification Report (KNN): precision recall f1-score support 0 0.62 0.59 0.60 68 1 0.78 0.80 0.79 122 accuracy 0.72 190 macro avg 0.70 0.69 0.69 190 weighted avg 0.72 0.72 0.72 190

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
K-Nearest Neighbors (KNN)	The K-Nearest Neighbors (KNN) algorithm was selected as the final modelforpredictinglivercirrhosisduetoitsimpressiveperformance metrics and suitability for the problem at hand. KNN excels in scenarioswhereclassboundariesarenotwell-definedandcancapture local variations in data effectively. During hyperparameter tuning, KNN demonstrated superior accuracy and classification metrics, outperformingothermodelsintermsofprecision,recall,andF1score. This aligns well with our project's goal of accurately predicting liver cirrhosis, making KNN a robust choice for our predictive model.