

Model Optimization and Tuning Phase Template

Date	28th June 2025
Team ID	LTVIP2025TMID41511
Project Title	Revolutionizing Liver Care: Predicting Liver Cirrhosis Using Advanced Machine Learning Techniques.
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):

Model	Tuned Hyperparameters	Optimal Values
Naive Bayes	No hyperparameters to tune for GaussianNB, directly fitting and scoring	<div>Train score: 0.8353096179183136 Test score: 0.7789473684210526 Accuracy on test set: 0.7789473684210526</div>
Random Forest	<pre>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] }</pre>	<div><pre>print('Best Hyperparameters for Random Forest:', rf_best_params) print('Train score:', rf_train_score) print('Test score:', rf_test_score)</pre></div> <div>✓ 62s X parameters + log</div> <div>Best Hyperparameters for Random Forest: {'n_estimators': 400, 'min_samples_split': 10, Train score: 0.930171277997365 Test score: 0.8736842105263158</div>

Logistic Regression CV	Logistic Regression CV automatically handles hyperparameter tuning with cross-validation	<hr/> Initial Train score: 0.8840579710144928 Initial Test score: 0.8157894736842105
Ridge Classifier	<pre> # 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_ </pre>	Optimal hyperparameters for Ridge Classifier: {'alpha': 100} Accuracy on test set: 0.8210526315789474
Support Vector Classifier	<pre> # 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_ </pre>	<hr/> Accuracy on test set: 0.64 Initial Train score: 0.7127799736495388 Initial Test score: 0.6421052631578947
Logistic Regression	<pre> # 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) </pre>	Optimal hyperparameters for Logistic Regression: {'C': 0.01, 'penalty': 'l2'} Accuracy on test set: 0.8052631578947368
XG Boost	<pre> # 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_ </pre>	<hr/> Initial Train score: 0.9920948616600791 Initial Test score: 0.8421052631578947 Accuracy on test set: 0.84

KNN	<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>	<div>Train score with tuned model: 0.8089591567852438</div> <div>Test score with tuned model: 0.7210526315789474</div> <div>Optimal hyperparameters for KNN: {'n_neighbors': 21}</div> <div>Accuracy on test set: 0.72</div>
-----	---	--

Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric																														
Naive Bayes	<div>Confusion Matrix (Naive Bayes):</div> <div>[[49 19]</div> <div>[23 99]]</div> <div>Classification Report (Naive Bayes):</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.68</td><td>0.72</td><td>0.70</td><td>68</td></tr><tr><td>1</td><td>0.84</td><td>0.81</td><td>0.82</td><td>122</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.78</td><td>190</td></tr><tr><td>macro avg</td><td>0.76</td><td>0.77</td><td>0.76</td><td>190</td></tr><tr><td>weighted avg</td><td>0.78</td><td>0.78</td><td>0.78</td><td>190</td></tr></table>		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
	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																											
Random Forest	<div>Confusion Matrix (Random Forest):</div> <div>[[51 17]</div> <div>[8 114]]</div> <div>Classification Report (Random Forest):</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.86</td><td>0.75</td><td>0.80</td><td>68</td></tr><tr><td>1</td><td>0.87</td><td>0.93</td><td>0.90</td><td>122</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>190</td></tr><tr><td>macro avg</td><td>0.87</td><td>0.84</td><td>0.85</td><td>190</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.87</td><td>0.87</td><td>190</td></tr></table>		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
	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																											
Logistic Regression CV	<div>Confusion Matrix (Logistic Regression CV):</div> <div>[[43 25]</div> <div>[10 112]]</div> <div>Classification Report (Logistic Regression CV):</div> <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0</td><td>0.81</td><td>0.63</td><td>0.71</td><td>68</td></tr><tr><td>1</td><td>0.82</td><td>0.92</td><td>0.86</td><td>122</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>190</td></tr><tr><td>macro avg</td><td>0.81</td><td>0.78</td><td>0.79</td><td>190</td></tr><tr><td>weighted avg</td><td>0.82</td><td>0.82</td><td>0.81</td><td>190</td></tr></table>		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
	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																											

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 0.82 190 weighted avg 0.82 190 </pre>
Support Vector 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 0.58 190 weighted avg 0.60 190 </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 0.80 190 weighted avg 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 0.84 190 weighted avg 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)	<p>The K-Nearest Neighbors (KNN) algorithm was selected as the final model for predicting liver cirrhosis due to its impressive performance metrics and suitability for the problem at hand. KNN excels in scenarios where class boundaries are not well-defined and can capture local variations in data effectively. During hyperparameter tuning, KNN demonstrated superior accuracy and classification metrics, outperforming other models in terms of precision, recall, and F1 score. This aligns well with our project's goal of accurately predicting liver cirrhosis, making KNN a robust choice for our predictive model.</p>