

## Model Optimization and Tuning Phase Report

Date	9 March 2025
Skillwallet ID	SWUID20250188620
Project Title	Anemia Sense: Leveraging Machine Learning for Precise Anemia Recognition
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
Logistic Regression	<pre># Define the parameter grid for Logistic Regression param_grid = {     'C': [0.1, 1, 10, 100],     'penalty': ['l1', 'l2'],     'solver': ['liblinear'] # liblinear supports both l1 and l2 }  # Create and configure the GridSearchCV object grid_search = GridSearchCV(estimator=LogisticRegression(),                            param_grid=param_grid,                            cv=5,                            scoring='accuracy',                            verbose=1,                            n_jobs=-1)</pre>	<pre># Fit the model to find the best parameters print("Tuning Logistic Regression...") grid_search.fit(x_train_balanced, y_train_balanced)  # Print the best findings print("\nBest Parameters found:", grid_search.best_params_) print("Best Accuracy through Cross-Validation: {:.2f}%".format(grid_search.best_score_ * 100))  Tuning Logistic Regression... Fitting 5 folds for each of 8 candidates, totalling 40 fits  Best Parameters found: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'} Best Accuracy through Cross-Validation: 99.29%</pre>
Random Forest	<pre>param_grid = {     'n_estimators': [50, 100, 200],     'max_depth': [10, 20, None],     'min_samples_split': [2, 5, 10],     'min_samples_leaf': [1, 2, 4] }  # Create and configure the GridSearchCV object grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),                            param_grid=param_grid,                            cv=5,                            scoring='accuracy',                            verbose=1,                            n_jobs=-1)</pre>	<pre># Fit the model to find the best parameters print("Tuning Random Forest Classifier...") grid_search.fit(x_train_balanced, y_train_balanced)  # Print the best findings print("\nBest Parameters found:", grid_search.best_params_) print("Best Accuracy through Cross-Validation: {:.2f}%".format(grid_search.best_score_ * 100))  Tuning Random Forest Classifier... Fitting 5 folds for each of 11 candidates, totalling 405 fits  Best Parameters found: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50} Best Accuracy through Cross-Validation: 100.00%</pre>

Naïve Bayes	<pre># Define the parameter grid for Gaussian Naive Bayes param_grid = {     'var_smoothing': np.logspace(0, -9, num=100) }  # Create and configure the GridSearchCV object grid_search = GridSearchCV(estimator=GaussianNB(),                            param_grid=param_grid,                            cv=5,                            scoring='accuracy',                            verbose=1,                            n_jobs=-1)</pre>	<pre># Fit the model to find the best parameters print("Tuning Gaussian Naive Bayes...") grid_search.fit(x_train_balanced, y_train_balanced)  # Print the best findings print("\nBest Parameters found:", grid_search.best_params_) print("Best Accuracy through Cross-Validation: {:.2f}%".format(grid_search.best_score_ * 100))  Tuning Gaussian Naive Bayes... Fitting 5 folds for each of 100 candidates, totalling 500 fits  Best Parameters found: {'var_smoothing': np.float64(0.0002310129700083158)} Best Accuracy through Cross Validation: 94.21%</pre>
SVM	<pre># Define the parameter grid for SVM param_grid = {     'C': [0.1, 1, 10, 100],     'gamma': [1, 0.1, 0.01, 0.001],     'kernel': ['rbf', 'linear'] }  # Create and configure the GridSearchCV object grid_search = GridSearchCV(estimator=SVC(),                            param_grid=param_grid,                            cv=5,                            scoring='accuracy',                            verbose=1,                            n_jobs=-1)</pre>	<pre># Fit the model to find the best parameters print("Tuning Support Vector Machine...") grid_search.fit(x_train_balanced, y_train_balanced)  # Print the best findings print("\nBest Parameters found:", grid_search.best_params_) print("Best Accuracy through Cross-Validation: {:.2f}%".format(grid_search.best_score_ * 100))  Tuning Support Vector Machine... Fitting 5 folds for each of 32 candidates, totalling 160 fits  Best Parameters found: {'C': 100, 'gamma': 1, 'kernel': 'linear'} Best Accuracy through Cross Validation: 99.00%</pre>
Gradient Boosting	<pre># Define the parameter grid for Gradient Boosting param_grid = {     'n_estimators': [50, 100, 150],     'learning_rate': [0.01, 0.1, 0.2],     'max_depth': [3, 4, 5] }  # Create and configure the GridSearchCV object grid_search = GridSearchCV(estimator=GradientBoostingClassifier(random_state=42),                            param_grid=param_grid,                            cv=5,                            scoring='accuracy',                            verbose=1,                            n_jobs=-1)</pre>	<pre># Fit the model to find the best parameters print("Tuning Gradient Boosting Classifier...") grid_search.fit(x_train_balanced, y_train_balanced)  # Print the best findings print("\nBest Parameters found:", grid_search.best_params_) print("Best Accuracy through Cross-Validation: {:.2f}%".format(grid_search.best_score_ * 100))  Tuning Gradient Boosting Classifier... Fitting 5 folds for each of 27 candidates, totalling 135 fits  Best Parameters found: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50} Best Accuracy through Cross Validation: 100.00%</pre>

## Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric																														
Logistic Regression	<pre>print("Classification Report") print(clr)</pre> <div><p>Accuracy Score</p><p>0.9824561403508771</p><p>Classification Report</p><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>1.00</td><td>0.97</td><td>0.98</td><td>157</td></tr><tr><td>1</td><td>0.96</td><td>1.00</td><td>0.98</td><td>128</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.98</td><td>285</td></tr><tr><td>macro avg</td><td>0.98</td><td>0.98</td><td>0.98</td><td>285</td></tr><tr><td>weighted avg</td><td>0.98</td><td>0.98</td><td>0.98</td><td>285</td></tr></tbody></table></div> <pre>print("Confusion Matrix") print(confusion_matrix(y_test, y_pred_log))</pre> <div><p>Confusion Matrix</p><pre>[[152  5]  [  0 128]]</pre></div>		precision	recall	f1-score	support	0	1.00	0.97	0.98	157	1	0.96	1.00	0.98	128	accuracy			0.98	285	macro avg	0.98	0.98	0.98	285	weighted avg	0.98	0.98	0.98	285
	precision	recall	f1-score	support																											
0	1.00	0.97	0.98	157																											
1	0.96	1.00	0.98	128																											
accuracy			0.98	285																											
macro avg	0.98	0.98	0.98	285																											
weighted avg	0.98	0.98	0.98	285																											

Random Forest	<pre>print("Classification Report") print(classification_report(y_test, y_pred_rf))</pre> <table><tr><td colspan="5">Accuracy Score</td></tr><tr><td colspan="5">0.9964912280701754</td></tr><tr><td colspan="5">Classification Report</td></tr><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>0</td><td>1.00</td><td>0.99</td><td>1.00</td><td>157</td></tr><tr><td>1</td><td>0.99</td><td>1.00</td><td>1.00</td><td>128</td></tr><tr><td>accuracy</td><td></td><td></td><td>1.00</td><td>285</td></tr><tr><td>macro avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>285</td></tr><tr><td>weighted avg</td><td>1.00</td><td>1.00</td><td>1.00</td><td>285</td></tr></table> <pre>print("Confusion Matrix") print(confusion_matrix(y_test, y_pred_rf))</pre> <table><tr><td colspan="2">Confusion Matrix</td></tr><tr><td>[[156  1]</td><td></td></tr><tr><td>[ 0 128]]</td><td></td></tr></table>	Accuracy Score					0.9964912280701754					Classification Report						precision	recall	f1-score	support	0	1.00	0.99	1.00	157	1	0.99	1.00	1.00	128	accuracy			1.00	285	macro avg	1.00	1.00	1.00	285	weighted avg	1.00	1.00	1.00	285	Confusion Matrix		[[156  1]		[ 0 128]]	
Accuracy Score																																																				
0.9964912280701754																																																				
Classification Report																																																				
	precision	recall	f1-score	support																																																
0	1.00	0.99	1.00	157																																																
1	0.99	1.00	1.00	128																																																
accuracy			1.00	285																																																
macro avg	1.00	1.00	1.00	285																																																
weighted avg	1.00	1.00	1.00	285																																																
Confusion Matrix																																																				
[[156  1]																																																				
[ 0 128]]																																																				
Naïve Bayes	<pre>print("Classification Report") print(classification_report(y_test, y_pred_gnb))</pre> <table><tr><td colspan="5">Accuracy Score</td></tr><tr><td colspan="5">0.968421052631579</td></tr><tr><td colspan="5">Classification Report</td></tr><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>0</td><td>0.99</td><td>0.96</td><td>0.97</td><td>157</td></tr><tr><td>1</td><td>0.95</td><td>0.98</td><td>0.97</td><td>128</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.97</td><td>285</td></tr><tr><td>macro avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td>285</td></tr><tr><td>weighted avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td>285</td></tr></table> <pre>print("Confusion Matrix") print(confusion_matrix(y_test, y_pred_gnb))</pre> <table><tr><td colspan="2">Confusion Matrix</td></tr><tr><td>[[150  7]</td><td></td></tr><tr><td>[ 2 126]]</td><td></td></tr></table>	Accuracy Score					0.968421052631579					Classification Report						precision	recall	f1-score	support	0	0.99	0.96	0.97	157	1	0.95	0.98	0.97	128	accuracy			0.97	285	macro avg	0.97	0.97	0.97	285	weighted avg	0.97	0.97	0.97	285	Confusion Matrix		[[150  7]		[ 2 126]]	
Accuracy Score																																																				
0.968421052631579																																																				
Classification Report																																																				
	precision	recall	f1-score	support																																																
0	0.99	0.96	0.97	157																																																
1	0.95	0.98	0.97	128																																																
accuracy			0.97	285																																																
macro avg	0.97	0.97	0.97	285																																																
weighted avg	0.97	0.97	0.97	285																																																
Confusion Matrix																																																				
[[150  7]																																																				
[ 2 126]]																																																				
SVM	<pre>print("Classification Report") print(classification_report(y_test, y_pred_svm))</pre> <table><tr><td colspan="5">Accuracy Score</td></tr><tr><td colspan="5">0.9228070175438596</td></tr><tr><td colspan="5">Classification Report</td></tr><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>0</td><td>1.00</td><td>0.86</td><td>0.92</td><td>157</td></tr><tr><td>1</td><td>0.85</td><td>1.00</td><td>0.92</td><td>128</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.92</td><td>285</td></tr><tr><td>macro avg</td><td>0.93</td><td>0.93</td><td>0.92</td><td>285</td></tr><tr><td>weighted avg</td><td>0.93</td><td>0.92</td><td>0.92</td><td>285</td></tr></table> <pre>print("Confusion Matrix") print(confusion_matrix(y_test, y_pred_svm))</pre> <table><tr><td colspan="2">Confusion Matrix</td></tr><tr><td>[[135  22]</td><td></td></tr><tr><td>[ 0 128]]</td><td></td></tr></table>	Accuracy Score					0.9228070175438596					Classification Report						precision	recall	f1-score	support	0	1.00	0.86	0.92	157	1	0.85	1.00	0.92	128	accuracy			0.92	285	macro avg	0.93	0.93	0.92	285	weighted avg	0.93	0.92	0.92	285	Confusion Matrix		[[135  22]		[ 0 128]]	
Accuracy Score																																																				
0.9228070175438596																																																				
Classification Report																																																				
	precision	recall	f1-score	support																																																
0	1.00	0.86	0.92	157																																																
1	0.85	1.00	0.92	128																																																
accuracy			0.92	285																																																
macro avg	0.93	0.93	0.92	285																																																
weighted avg	0.93	0.92	0.92	285																																																
Confusion Matrix																																																				
[[135  22]																																																				
[ 0 128]]																																																				

## Gradient Boosting

```
print("Classification Report")
print(classification_report(y_test, y_pred_gb))
```

Accuracy Score  
1.0

Classification Report		precision	recall	f1-score	support
0	1.00	1.00	1.00	1.00	157
1	1.00	1.00	1.00	1.00	128
accuracy				1.00	285
macro avg		1.00	1.00	1.00	285
weighted avg		1.00	1.00	1.00	285

```
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred_gb))
```

Confusion Matrix  
[[157 0]  
[ 0 128]]

**Final Model Selection Justification (2 Marks):**

Final Model	Reasoning
Gradient Boosting	The Gradient Boosting model was selected for its superior performance, exhibiting high accuracy during hyperparameter tuning. Its ability to handle complex relationships, minimize overfitting, and optimize predictive accuracy aligns with project objectives, justifying its selection as the final model.