+	Model Optimization and Tuning Phase Report					
	Date	19 May 2025				
	Team ID	SWTID1750233055				
	Project Title	SmartLender - Applicant Credibility Prediction for Loan Approval				
	Maximum Marks	10 Marks				

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase focuses on improving the performance of machine learning models by refining their configurations. This stage ensures that the chosen algorithm not only performs well on the training data but also generalizes effectively to unseen data.

Model	Grid Search	Optimal Values
Decision Tree	<pre>param_grid = { 'n_estimators': [10, 25, 50, 75, 100], 'learning_rate': [0.5, 1.0, 1.5], # GridSearchCV setup grid = GridSearchCV(estimator=abc, param_grid=param_grid, cv=5, n_jobs=-1) grid.fit(X_train, y_train)</pre>	<pre># Best model print("Best Parameters:", grid.best_params_) # Evaluate y_pred_grid = grid.predict(X_test) print("Grid Search Accuracy:", accuracy_score(y_test, y_pred_grid))</pre>
		Best Parameters: {'learning_rate': 0.5, 'n_estimators': 75} Grid Search Accuracy: 0.7553191489361702

```
Ada
               param_grid = {
                                                                                             # Best model
Boost
                   'n estimators': [10, 25, 50, 75, 100],
                                                                                            print("Best Parameters:", grid.best params )
                   'learning rate': [0.5, 1.0, 1.5],
                                                                                            # Evaluate
                                                                                            y pred grid = grid.predict(X test)
               # GridSearchCV setup
                                                                                             print("Grid Search Accuracy:", accuracy score(y test, y pred grid))
               grid = GridSearchCV(estimator=abc, param grid=param grid, cv=5, n_jobs=-1)
               grid.fit(X_train, y_train)
                                                                                             Best Parameters: {'learning rate': 0.5, 'n estimators': 75}
                                                                                             Grid Search Accuracy: 0.7553191489361702
KNN
                param grid = {
                                                                                            # Best model
                    'n estimators': [10, 25, 50, 75, 100],
                                                                                            print("Best Parameters:", grid.best params )
                    'learning rate': [0.5, 1.0, 1.5],
                                                                                            # Fvaluate
                                                                                            y pred grid = grid.predict(X test)
                # GridSearchCV setup
                                                                                            print("Grid Search Accuracy:", accuracy score(y test, y pred grid))
                grid = GridSearchCV(estimator=abc, param_grid=param_grid, cv=5, n jobs=-1)
                grid.fit(X train, y train)
                                                                                            Best Parameters: {'learning rate': 0.5, 'n estimators': 75}
                                                                                            Grid Search Accuracy: 0.7553191489361702
Gradient
                param_grid = {
                                                                                            # Best model
Boosting
                    'n estimators': [10, 25, 50, 75, 100],
                                                                                            print("Best Parameters:", grid.best params )
                    'learning_rate': [0.5, 1.0, 1.5],
                                                                                            # Evaluate
                                                                                            y pred grid = grid.predict(X test)
                # GridSearchCV setup
                                                                                            print("Grid Search Accuracy:", accuracy_score(y_test, y_pred_grid))
                grid = GridSearchCV(estimator=abc, param_grid=param_grid, cv=5, n_jobs=-1)
                grid.fit(X_train, y_train)
                                                                                             Best Parameters: {'learning rate': 0.5, 'n estimators': 75}
                                                                                            Grid Search Accuracy: 0.7553191489361702
```

Performance Metrics Comparison Report (2 Marks):

Model		Optimized Metric				
Decision Tree dec=DecisionTreeClassifier(random_state=49) base = DecisionTreeClassifier(max_depth=3) dec.fit(X_train,y_train) pred_abc = dec.predict(X_test) print(classification_report(y_test,pred_abc)) print(confusion_matrix(y_test,pred_abc))						
		precision	recall	f1-score	support	
	0	0.68	0.66	0.67	184	
	1	0.68	0.70	0.69	192	
	accuracy			0.68	376	
	macro avg	0.68	0.68	0.68	376	
	weighted avg	0.68	0.68	0.68	376	
	[[121 63] [58 134]]					

```
Ada Boost
                        abc = AdaBoostClassifier(random_state=99)
                        base = DecisionTreeClassifier(max_depth=3)
                        abc.fit(X_train,y_train)
                        pred_abc = abc.predict(X_test)
                        print(classification_report(y_test,pred_abc))
                        print(confusion_matrix(y_test,pred_abc))
                        C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_w
                        nd will be removed in 1.6. Use the SAMME algorithm to circumve
                        warnings.warn(
                                      precision
                                                   recall f1-score
                                                                      support
                                           0.72
                                                     0.75
                                                               0.74
                                   0
                                                                          184
                                           0.75
                                                     0.72
                                                               0.74
                                                                          192
                                                               0.74
                                                                          376
                            accuracy
                                           0.74
                                                     0.74
                                                               0.74
                                                                          376
                           macro avg
                        weighted avg
                                           0.74
                                                     0.74
                                                               0.74
                                                                          376
                         [138 46]
```

KNN	1 ICN - 4 - 1- 1						
KININ	kn=KNeighborsClassifier()						
	knn base = DecisionTreeClassifier(max_depth=3)						
	kn.fit(X_train,y_train)						
	pred_abc = kn.						
	<pre>print(classification_report(y_test,pred_abc))</pre>						
	print(confusio	c(confusion_matrix(y_test,pred_abc))					
		precision	recall	f1-score	support		
	0	0.65	0.70	0.67	184		
	1	0.69	0.65	0.67	192		
	accuracy			0.67	376		
	macro avg	0.67	0.67	0.67	376		
	weighted avg	0.67	0.67	0.67	376		
	[[128 56]						
	[68 124]]						
Gradient Boosting	<pre>gb=GradientBoo base = Decisio gb.fit(X_train pred abc = gb.</pre>	nTreeClassi ,y_train)	fier(max_	_	9)		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb.</pre>	nTreeClassi ,y_train) predict(X_t	fier(max_ est)	_depth=3)	·		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train</pre>	nTreeClassi ,y_train) predict(X_t cation_repo	fier(max_ est) rt(y_test	_depth=3) :,pred_abc)	·		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio</pre>	nTreeClassi ,y_train) predict(X_t cation_repo	fier(max_ est) rt(y_test test,pred	_depth=3) :,pred_abc)	·		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision	fier(max_ est) rt(y_test test,pred recall	_depth=3) :,pred_abc) _abc)) f1-score) support		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_	fier(max_ est) rt(y_test test,pred	_depth=3) c,pred_abc) l_abc)))		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73	fier(max_ est) rt(y_test test,pred recall 0.76	depth=3) .,pred_abc) l_abc)) f1-score 0.74	support		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73	fier(max_ est) rt(y_test test,pred recall 0.76	depth=3) .,pred_abc) l_abc)) f1-score 0.74	support		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio)</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73	fier(max_ est) rt(y_test test,pred recall 0.76	depth=3) (,pred_abc) (_abc)) f1-score 0.74 0.74	support 184 192		
Gradient Boosting	<pre>base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio) 0 1 accuracy</pre>	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73 0.76	fier(max_ est) rt(y_test test,prec recall 0.76 0.72	depth=3) [,pred_abc) [_abc)) f1-score 0.74 0.74	support 184 192 376		
Gradient Boosting	base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio 0 1 accuracy macro avg weighted avg	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73 0.76	fier(max_ est) rt(y_test test,pred recall 0.76 0.72	(,pred_abc) (,pred_abc) (_abc)) f1-score 0.74 0.74 0.74	support 184 192 376 376		
Gradient Boosting	base = Decisio gb.fit(X_train pred_abc = gb. print(classifi print(confusio 0 1 accuracy macro avg	nTreeClassi ,y_train) predict(X_t cation_repo n_matrix(y_ precision 0.73 0.76	fier(max_ est) rt(y_test test,pred recall 0.76 0.72	(,pred_abc) (,pred_abc) (_abc)) f1-score 0.74 0.74 0.74	support 184 192 376 376		

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Ada Boost	The AdaBoost model was selected for its superior performance, exhibiting high accuracy during hyperparameter tuning. Its ability to focus on difficult-to-classify instances, reduce variance, and maintain strong generalization makes it well-suited for this project. AdaBoost's effectiveness in handling imbalanced and noisy data, along with its lightweight nature and ease of integration into real-time systems, aligns with the project objectives — justifying its selection as the final model.