

Model Optimization and Tuning Phase Report

Date	21 June 2024
Team ID	739812
Project Title	Eudaimonia Engine: Machine Learning Delving into Happiness Classification
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
Decision Tree	<pre>#Hyperparameter Tuning for Decision Tree Model #Define Decision Tree Classifier dt=DecisionTreeClassifier() #Hyperparameter Tuning param_grid_dt = { 'criterion': ['gini', 'entropy'], 'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4] } grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt, cv=5, scoring='accuracy') grid_search_dt.fit(x_train, y_train) # Assuming you have x_train and y_train defined # Access best parameters for Decision Tree best_params_dt = grid_search_dt.best_params_ best_criterion = best_params_dt['criterion'] best_max_depth = best_params_dt['max_depth'] best_min_samples_split = best_params_dt['min_samples_split'] best_min_samples_leaf = best_params_dt['min_samples_leaf'] # Create the tuple best_param = (best_criterion, best_max_depth, best_min_samples_split, best_min_samples_leaf)</pre>	<pre>from sklearn.metrics import accuracy_score # Assuming you have defined and trained your classifier model classifier = dt classifier.fit(x_train, y_train) # Evaluate the performance of the tuned model y_pred = classifier.predict(x_test) accuracy = accuracy_score(y_test, y_pred) print(f'Optimal Hyperparameters: {best_param}') print(f'Accuracy on test set: {accuracy}')</pre> <p>Optimal Hyperparameters: ('entropy', None, 10, 1) Accuracy on test set: 0.7241379310344828</p>
Random Forest	<pre>#Hyperparameter Tuning for Random Forest Model #Define Random forest Tree Classifier rf = RandomForestClassifier() #Hyperparameter Tuning # Define the parameter grid for hyperparameter tuning param_grid = { 'n_estimators': [50, 100, 200], 'criterion': ['gini', 'entropy'], 'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4] }</pre>	<pre>from sklearn.metrics import accuracy_score # Assuming you have defined and trained your classifier model classifier = rf classifier.fit(x_train, y_train) # Evaluate the performance of the tuned model y_pred = classifier.predict(x_test) accuracy = accuracy_score(y_test, y_pred) print(f'Optimal Hyperparameters: {best_param}') print(f'Accuracy on test set: {accuracy}')</pre> <p>Optimal Hyperparameters: ('entropy', None, 10, 1) Accuracy on test set: 0.5862068965517241</p>

KNN	<pre>#Hyperparameter Tuning For KNN Model from sklearn.model_selection import GridSearchCV from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import accuracy_score # Define the KNN classifier knn = KNeighborsClassifier() # Define the hyperparameters to tune parameters = { 'n_neighbors': [3, 5, 7, 9], # Number of neighbors to consider 'weights': ['uniform', 'distance'], # Weight function used in prediction 'metric': ['euclidean', 'manhattan'] # Distance metric to use for the tree } # Perform grid search with cross-validation grid_search = GridSearchCV(knn, parameters, cv=5) grid_search.fit(x_train, y_train) # Get the best hyperparameters best_params = grid_search.best_params_ # Use the best model for prediction best_model = grid_search.best_estimator_ y_pred = best_model.predict(x_test)</pre>	<pre># Evaluate the performance of the tuned model accuracy = accuracy_score(y_test, y_pred) print(f'Optimal Hyperparameters: {best_params}') print(f'Accuracy on test set: {accuracy}')</pre> <p>Optimal Hyperparameters: {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'} Accuracy on test set: 0.5517241379310345</p>
SVC	<pre>#Hyperparameter Tuning For SVC Model from sklearn.model_selection import GridSearchCV from sklearn.svm import SVC from sklearn.metrics import accuracy_score # Define the SVC classifier svc = SVC() # Define the hyperparameters to tune parameters = { 'C': [0.1, 1, 10], # Regularization parameter 'kernel': ['linear', 'rbf'], # Kernel type 'gamma': ['scale', 'auto'] # Kernel coefficient } # Perform grid search with cross-validation grid_search = GridSearchCV(svc, parameters, cv=5) grid_search.fit(x_train, y_train) # Get the best hyperparameters best_params = grid_search.best_params_ # Use the best model for prediction best_model = grid_search.best_estimator_ y_pred = best_model.predict(x_test)</pre>	<pre># Evaluate the performance of the tuned model accuracy = accuracy_score(y_test, y_pred) print(f'Optimal Hyperparameters: {best_params}') print(f'Accuracy on test set: {accuracy}')</pre> <p>Optimal Hyperparameters: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'} Accuracy on test set: 0.4827586206896552</p>
Logistic Model	<pre>#Hyperparameter Tuning For Logistic Model from sklearn.model_selection import GridSearchCV from sklearn.linear_model import LogisticRegression from sklearn.metrics import accuracy_score # Define the Logistic Regression classifier log_reg = LogisticRegression() # Define the hyperparameters to tune parameters = { 'penalty': ['l1', 'l2'], # Regularization type 'C': [0.1, 0.5, 1, 2, 5, 10], # Inverse of regularization strength 'solver': ['liblinear', 'saga'], # Optimization algorithm 'max_iter': [100, 200, 300] # Maximum number of iterations } # Perform grid search with cross-validation grid_search = GridSearchCV(log_reg, parameters, cv=5) grid_search.fit(x_train, y_train) # Get the best hyperparameters best_params = grid_search.best_params_ # Use the best model for prediction best_model = grid_search.best_estimator_ y_pred = best_model.predict(x_test)</pre>	<pre># Evaluate the performance of the tuned model accuracy = accuracy_score(y_test, y_pred) print(f'Optimal Hyperparameters: {best_params}') print(f'Accuracy on test set: {accuracy}')</pre> <p>Optimal Hyperparameters: {'C': 2, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'} Accuracy on test set: 0.4827586206896552</p>

Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric
Decision Tree	<pre>#Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr)</pre> <pre> precision recall f1-score support 0 0.73 0.57 0.64 14 1 0.67 0.80 0.73 15 accuracy 0.69 macro avg 0.70 weighted avg 0.70</pre> <pre>#Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm)</pre> <pre>[[8 6] [3 12]]</pre>

Random Forest	<pre>#Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr)</pre> <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>0.56</td><td>0.36</td><td>0.43</td><td>14</td></tr><tr><td>1</td><td>0.55</td><td>0.73</td><td>0.63</td><td>15</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.55</td><td>29</td></tr><tr><td>macro avg</td><td>0.55</td><td>0.55</td><td>0.53</td><td>29</td></tr><tr><td>weighted avg</td><td>0.55</td><td>0.55</td><td>0.54</td><td>29</td></tr></tbody></table> <pre>#Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm)</pre> <pre>[[5 9] [4 11]]</pre>		precision	recall	f1-score	support	0	0.56	0.36	0.43	14	1	0.55	0.73	0.63	15	accuracy			0.55	29	macro avg	0.55	0.55	0.53	29	weighted avg	0.55	0.55	0.54	29
	precision	recall	f1-score	support																											
0	0.56	0.36	0.43	14																											
1	0.55	0.73	0.63	15																											
accuracy			0.55	29																											
macro avg	0.55	0.55	0.53	29																											
weighted avg	0.55	0.55	0.54	29																											
KNN	<pre>#Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr)</pre> <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>0.25</td><td>0.14</td><td>0.18</td><td>14</td></tr><tr><td>1</td><td>0.43</td><td>0.60</td><td>0.50</td><td>15</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.38</td><td>29</td></tr><tr><td>macro avg</td><td>0.34</td><td>0.37</td><td>0.34</td><td>29</td></tr><tr><td>weighted avg</td><td>0.34</td><td>0.38</td><td>0.35</td><td>29</td></tr></tbody></table> <pre>#Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm)</pre> <pre>[[2 12] [6 9]]</pre>		precision	recall	f1-score	support	0	0.25	0.14	0.18	14	1	0.43	0.60	0.50	15	accuracy			0.38	29	macro avg	0.34	0.37	0.34	29	weighted avg	0.34	0.38	0.35	29
	precision	recall	f1-score	support																											
0	0.25	0.14	0.18	14																											
1	0.43	0.60	0.50	15																											
accuracy			0.38	29																											
macro avg	0.34	0.37	0.34	29																											
weighted avg	0.34	0.38	0.35	29																											
SVC	<pre>#Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr)</pre> <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>0.25</td><td>0.14</td><td>0.18</td><td>14</td></tr><tr><td>1</td><td>0.43</td><td>0.60</td><td>0.50</td><td>15</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.38</td><td>29</td></tr><tr><td>macro avg</td><td>0.34</td><td>0.37</td><td>0.34</td><td>29</td></tr><tr><td>weighted avg</td><td>0.34</td><td>0.38</td><td>0.35</td><td>29</td></tr></tbody></table> <pre>#Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm)</pre> <pre>[[2 12] [6 9]]</pre>		precision	recall	f1-score	support	0	0.25	0.14	0.18	14	1	0.43	0.60	0.50	15	accuracy			0.38	29	macro avg	0.34	0.37	0.34	29	weighted avg	0.34	0.38	0.35	29
	precision	recall	f1-score	support																											
0	0.25	0.14	0.18	14																											
1	0.43	0.60	0.50	15																											
accuracy			0.38	29																											
macro avg	0.34	0.37	0.34	29																											
weighted avg	0.34	0.38	0.35	29																											
Logistic Model	<pre>#Classification Report from sklearn.metrics import classification_report cr=classification_report(y_test,y_pred) print(cr)</pre> <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>0.25</td><td>0.14</td><td>0.18</td><td>14</td></tr><tr><td>1</td><td>0.43</td><td>0.60</td><td>0.50</td><td>15</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.38</td><td>29</td></tr><tr><td>macro avg</td><td>0.34</td><td>0.37</td><td>0.34</td><td>29</td></tr><tr><td>weighted avg</td><td>0.34</td><td>0.38</td><td>0.35</td><td>29</td></tr></tbody></table> <pre>#Confusion Matrix from sklearn.metrics import confusion_matrix cm=confusion_matrix(y_test,y_pred) print(cm)</pre> <pre>[[2 12] [6 9]]</pre>		precision	recall	f1-score	support	0	0.25	0.14	0.18	14	1	0.43	0.60	0.50	15	accuracy			0.38	29	macro avg	0.34	0.37	0.34	29	weighted avg	0.34	0.38	0.35	29
	precision	recall	f1-score	support																											
0	0.25	0.14	0.18	14																											
1	0.43	0.60	0.50	15																											
accuracy			0.38	29																											
macro avg	0.34	0.37	0.34	29																											
weighted avg	0.34	0.38	0.35	29																											

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Decision Tree Model	The Decision Tree 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.