Model Optimization and Tuning Phase Template

Date	8 July 2024
Team ID	SWTID1720082525
Project Title	Early Prediction of Chronic Kidney Disease Using Machine Learning
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	from sklearn.linear_model import LogisticRegression
	from sklearn.model_selection import GridSearchCV
	
	lr_classifier = LogisticRegression(random_state=123)
	
	# Define the hyperparameters and their possible values for tuning
	param_grid = {
	    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
	    'C': [0.001, 0.01, 0.1, 1, 10, 100],
	    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
	    'max_iter': [100, 200, 300, 500]
	}
		# Set up GridSearchCV
		grid_search = GridSearchCV(estimator=lr_classifier, param_grid=param_grid, cv=5, scoring='f1', n_jobs=-1)
		
		grid_search.fit(X_train, y_train)
		
		# Evaluate the performance of the tuned model
		f1_score = grid_search.score(X_test, y_test)
		
		# Print results
		print(f"Optimal Hyperparameters: {grid_search.best_params_}")
		print(f"F1 Score on Test Set: {f1_score}")
		
		Optimal Hyperparameters: {'C': 100, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
		F1 Score on Test Set: 0.9318181818181818

<table>
<tr>
<td>Decision
Tree
Regression</td>
<td><pre>
# Decison Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

dt_classifier = DecisionTreeClassifier(random_state=123)

param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
</pre></td>
<td><pre>
# Set up GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid, cv=5, scoring='f1', n_jobs=-1)

grid_search.fit(X_train, y_train)

f1_score = grid_search.score(X_test, y_test)

# Print results
print(f"Optimal Hyperparameters: {grid_search.best_params_}")
print(f"F1 Score on Test Set: {f1_score}")
</pre>
<pre>
Optimal Hyperparameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2}
F1 Score on Test Set: 0.9534883720930233
</pre></td>
</tr>
<tr>
<td>Random
Forest
Regression</td>
<td><pre>
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=123)

# Define the hyperparameters and their possible values for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}
</pre></td>
<td><pre>
# Set up GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='f1', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Evaluate the performance of the tuned model
f1_score = grid_search.score(X_test, y_test)

# Print results
print(f"Optimal Hyperparameters: {grid_search.best_params_}")
print(f"F1 Score on Test Set: {f1_score}")
</pre>
<pre>
Optimal Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1,
 'min_samples_split': 2, 'n_estimators': 200} F1 Score on Test Set: 0.9761904761904762
</pre></td>
</tr>
<tr>
<td>KNN</td>
<td><pre>
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
knn_classifier = KNeighborsClassifier()

# Define the hyperparameters and their possible values for tuning
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}
</pre></td>
<td><pre>
# Set up GridSearchCV
grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid, cv=5, scoring='f1', n_jobs=-1)

grid_search.fit(X_train, y_train)

# Evaluate the performance of the tuned model
f1_score = grid_search.score(X_test, y_test)

# Print results
print(f"Optimal Hyperparameters: {grid_search.best_params_}")
print(f"F1 Score on Test Set: {f1_score}")
</pre>
<pre>
Optimal Hyperparameters: {'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
F1 Score on Test Set: 0.5925925925925926
</pre></td>
</tr>
</table>

Performance Metrics Comparison Report (2 Marks):

Model	Optimized Metric
Random Forest Regression	Classification Report for Random Forest: precision recall f1-score support 0 0.96 0.96 0.96 78 1 0.93 0.93 0.93 42 accuracy 0.95 120 macro avg 0.95 0.95 0.95 120 weighted avg 0.95 0.95 0.95 120 Confusion matrix of Random Forest [[76 2] [3 39]]
Decision Tree Regression	Classification Report for Decision Tree: precision recall f1-score support 0 0.95 0.92 0.94 78 1 0.86 0.90 0.88 42 accuracy 0.92 120 macro avg 0.91 0.91 0.91 120 weighted avg 0.92 0.92 0.92 120 Confusion Matrix for Decision Tree: [[72 6] [4 38]]
Logistic Regression	Classification Report for Logistic Regression: precision recall f1-score support 0 0.94 0.82 0.88 78 1 0.73 0.90 0.81 42 accuracy 0.85 120 macro avg 0.84 0.86 0.84 120 weighted avg 0.87 0.85 0.85 120 Confusion matrix of Logistic Regression [[64 14] [4 38]]

<table>
<tr>
<td>KNN</td>
<td>
<pre>
Classification Report for KNN:
              precision   recall f1-score   support

           0   0.75   0.54   0.63   78
           1   0.44   0.67   0.53   42

    accuracy   0.58   120
   macro avg   0.59   0.60   0.58   120
weighted avg   0.64   0.58   0.59   120
</pre>
<pre>
Confusion Matrix for KNN:
[[42 36]
 [14 28]]
</pre>
</td>
</tr>
</table>

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Random Forest Regression	The Random Forest model was chosen due to its impressive 95.8% accuracy achieved during hyperparameter tuning. Its ensemble method effectively manages complex data relationships and minimizes overfitting. Additionally, the model's capability to offer feature importance rankings, along with its strong performance, makes it well-suited for the project's goals of high predictive accuracy and interpretability.