Model Optimization and Tuning Phase Template

Date	10 July 2024
Team ID	SWTID1720082372
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	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)
		# 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': 0.001, 'max_iter': 100, 'penalty': 'none', 'solver': 'newton-cg'}
		F1 Score on Test Set: 0.9176470588235294

<table>
<tr>
<td>Decision Tree

Regression</td>
<td>
Decision Tree Regression
<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': 'entropy', 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5}
F1 Score on Test Set: 0.9397590361445782
</pre>
</td>
</tr>
<tr>
<td>Random Forest

Regression</td>
<td>
Random Forest Regression
<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': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 300}
F1 Score on Test Set: 0.951219512195122
</pre>
</td>
</tr>
<tr>
<td>KNN</td>
<td>
KNN
<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': 9, 'weights': 'distance'}
F1 Score on Test Set: 0.5607476635514019
</pre>
</td>
</tr>
</table>

Performance Metrics Comparison Report (2 Marks):

<table>
<tr><th>Model</th><th>Optimized Metric</th></tr>
<tr><td>Random Forest

Regression</td><td><pre>Classification Report for Random Forest:
              precision   recall f1-score   support

           0   0.96   0.99   0.97   78
           1   0.97   0.93   0.95   42

    accuracy   0.97   120
   macro avg   0.97   0.96   0.96   120
weighted avg   0.97   0.97   0.97   120

Confusion matrix of Random Forest
[[77 1]
 [3 39]]</pre></td></tr>
<tr><td>Decision Tree

Regression</td><td><pre>Classification Report for Decision Tree:
              precision   recall f1-score   support

           0   0.96   0.95   0.95   78
           1   0.91   0.93   0.92   42

    accuracy   0.94   120
   macro avg   0.93   0.94   0.94   120
weighted avg   0.94   0.94   0.94   120

Confusion Matrix for Decision Tree:
[[74 4]
 [3 39]]</pre></td></tr>
</table>

Logistic Regression

```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.96      0.88      0.92        78
           1       0.81      0.93      0.87        42

    accuracy                           0.90       120
   macro avg       0.89      0.91      0.89       120
weighted avg       0.91      0.90      0.90       120
```

```
Confusion matrix of Logistic Regression
[[69  9]
 [ 3 39]]
```

KNN

```
Classification Report for KNN:
              precision    recall  f1-score   support

           0       0.70      0.50      0.58        78
           1       0.39      0.60      0.47        42

    accuracy                           0.53       120
   macro avg       0.54      0.55      0.53       120
weighted avg       0.59      0.53      0.54       120
```

```
Confusion Matrix for KNN:
[[39 39]
 [17 25]]
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Random Forest Regression	The Random Forest model was selected for its outstanding 97% accuracy during hyperparameter tuning. Its ensemble approach excels at handling complex data relationships while reducing overfitting. The model's ability to provide feature importance rankings, coupled with its robust performance, aligns perfectly with the project's objectives for high predictive accuracy and interpretability.