Model Optimization and Tuning Phase

Date	11 July 2024
Team ID	SWTID1720013031
Project Title	Prediction and Analysis of Liver Patient Data 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
	`lr_param_grid = { 'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1.0, 10.0], 'solver': ['liblinear', 'saga'] }`
	`lr_s = LogisticRegression(max_iter=1000)`
		`lr_grid_search.fit(x_train_final, y_train)`
		`# Get best parameters`
		`lr_best_params = lr_grid_search.best_params_`
		`print("Best parameters for Logistic Regression:", lr_best_params)`
		`Fitting 5 folds for each of 20 candidates, totalling 100 fits`
		`Best parameters for Logistic Regression: {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'}`
		`lr_acc = accuracy_score(y_pred_lr, y_test)`
		`lr_acc`
		`0.6971428571428572`
		`LogisticRegression`
		`LogisticRegression(max_iter=1000, penalty='l1', solver='liblinear')`
		Best parameters for Logistic Regression: {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'}

Support Vector Classifier (SVC)	Support Vector Classifier (SVC)
	`svm_param_dist = { 'C': uniform(0.1, 10), 'kernel': ['linear', 'rbf'], 'gamma': ['scale', 'auto'] }`
	`svm_s = SVC()`
		`svm_best_params = svm_random_search.best_params_`
		`print("Best parameters for SVC:", svm_best_params)`
		`Best parameters for SVC: {'C': 7.41993941811405, 'gamma': 'scale', 'kernel': 'linear'}`
		`svm_acc = accuracy_score(y_pred_svm, y_test)`
		`svm_acc`
		`0.6971428571428572`
		SVC
		`SVC(C=7.41993941811405, kernel='linear')`
		Best parameters for SVC: {'C': 7.41993941811405, 'gamma': 'scale', 'kernel': 'linear'}

Random Forest Classifier	Random Forest Classifier
	`rfc_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] }`
	`rfc_s = RandomForestClassifier()`
		`rfc_best_params = rfc_grid_search.best_params_`
		`print("Best parameters for Random Forest Classifier:", rfc_best_params)`
		`Best parameters for Random Forest Classifier: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}`
		`rfc_acc = accuracy_score(ypred_rfc, y_test)`
		`rfc_acc`
		`0.6971428571428572`
		RandomForestClassifier
		`RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=10)`
		Best parameters for Random Forest Classifier: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}

K Neighbors Classifier	**K Neighbors Classifier**
	`knn_param_grid = { 'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform', 'distance'], 'metric': ['euclidean', 'manhattan'] }`
	`knn_s = KNeighborsClassifier()`
		`knn_best_params = knn_grid_search.best_params_`
		`print("Best parameters for K Neighbors Classifier:", knn_best_params)`
		`Best parameters for K Neighbors Classifier: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}`
		`knn_acc = accuracy_score(ypred_knn, y_test)`
		`knn_acc`
		`0.6628571428571428`
		`KNeighborsClassifier`
		`KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')`
		Best parameters for K Neighbors Classifier: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}

Performance Metrics Comparison Report (2 Marks):

Model	Baseline Metric	Optimized Metric

Logistic Regression	`lr = LogisticRegression(max_iter=1000)`
	`lr.fit(x_train_final, y_train)`
	`LogisticRegression`
	`LogisticRegression(max_iter=1000)`
	`y_pred_lr = lr.predict(x_test_final)`
	`lr_acc = accuracy_score(y_pred_lr, y_test)`
	`lr_acc`
	`0.6857142857142857`
	`print(classification_report(y_test, y_pred_lr))`
	precision recall f1-score support
	1 0.72 0.90 0.80 122
	2 0.45 0.19 0.27 53
	accuracy 0.69 175
	macro avg 0.59 0.55 0.53 175
	weighted avg 0.64 0.69 0.64 175
		Logistic Regression
		`lr_param_grid = { 'penalty': ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1.0, 10.0], 'solver': ['liblinear', 'saga'] }`
		`lr_s = LogisticRegression(max_iter=1000)`
		`LogisticRegression`
		`LogisticRegression(max_iter=1000, penalty='l1', solver='liblinear')`

```
lr_cm = confusion_matrix(y_pred_lr,y_test)
lr_cm
```

```
array([[110,  43],
       [ 12,  10]], dtype=int64)
```

```
lr_acc = accuracy_score(y_pred_lr, y_test)
lr_acc
```

```
0.6971428571428572
```

```
print(classification_report(y_test, y_pred_lr))
```

```
              precision    recall  f1-score   support

           1       0.72      0.93      0.81       122
           2       0.50      0.15      0.23        53

    accuracy                           0.70       175
   macro avg       0.61      0.54      0.52       175
weighted avg       0.65      0.70      0.64       175
```

```
lr_cm = confusion_matrix(y_pred_lr,y_test)
lr_cm
```

```
array([[114,  45],
       [  8,   8]], dtype=int64)
```

Support Vector Classifier (SVC)

```
svm = SVC()
svm.fit(x_train_final, y_train)
```

```
▾ SVC
SVC()
```

```
y_pred_svm = svm.predict(x_test_final)
```

```
svm_acc = accuracy_score(y_pred_svm, y_test)
svm_acc
```

```
0.6971428571428572
```

```
print(classification_report(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

           1       0.70      1.00      0.82       122
           2       0.00      0.00      0.00        53

    accuracy                           0.70       175
   macro avg       0.35      0.50      0.41       175
weighted avg       0.49      0.70      0.57       175
```

```
svm_cm = confusion_matrix(y_pred_svm,y_test)
svm_cm
```

```
array([[122,  53],
       [  0,   0]], dtype=int64)
```

Support Vector Classifier (SVC)

```
svm_param_dist = {
    'C': uniform(0.1, 10),
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```

```
svm = SVC(**svm_best_params)
```

```
svm.fit(x_train_final, y_train)
```

```
▾ SVC
SVC(C=7.41993941811405, kernel='linear')
```

```
y_pred_svm = svm.predict(x_test_final)
```

```
svm_acc = accuracy_score(y_pred_svm, y_test)
svm_acc
```

```
0.6971428571428572
```

```
print(classification_report(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

           1       0.70      1.00      0.82       122
           2       0.00      0.00      0.00        53

    accuracy                           0.70       175
   macro avg       0.35      0.50      0.41       175
weighted avg       0.49      0.70      0.57       175
```

```
svm_cm = confusion_matrix(y_pred_svm,y_test)
svm_cm
```

```
array([[122,  53],
       [  0,   0]], dtype=int64)
```

Support Vector Classifier (SVC)

<table>
<tr>
<td>Random Forest Classifier</td>
<td>

```
rfc = RandomForestClassifier()
rfc.fit(x_train_final, y_train)
```

RandomForestClassifier

RandomForestClassifier()

```
ypred_rfc = rfc.predict(x_test_final)
```

```
rfc_acc = accuracy_score(ypred_rfc, y_test)
rfc_acc
```

0.6857142857142857

```
print(classification_report(y_test, ypred_rfc))
```

	precision	recall	f1-score	support
1	0.74	0.85	0.79	122
2	0.47	0.30	0.37	53
accuracy			0.69	175
macro avg	0.60	0.58	0.58	175
weighted avg	0.66	0.69	0.66	175

```
rfc_cm = confusion_matrix(ypred_rfc,y_test)
rfc_cm
```

```
array([[104,  37],
       [ 18,  16]], dtype=int64)
```

</td>
<td>

Random Forest Classifier

```
rfc_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]
}
```

```
rfc_s = RandomForestClassifier()
```

RandomForestClassifier

RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=10)

```
ypred_rfc = rfc.predict(x_test_final)
```

```
rfc_acc = accuracy_score(ypred_rfc, y_test)
rfc_acc
```

0.6971428571428572

```
print(classification_report(y_test, ypred_rfc))
```

	precision	recall	f1-score	support
1	0.73	0.90	0.81	122
2	0.50	0.23	0.31	53
accuracy			0.70	175
macro avg	0.61	0.56	0.56	175
weighted avg	0.66	0.70	0.66	175

```
rfc_cm = confusion_matrix(ypred_rfc,y_test)
rfc_cm
```

```
array([[110,  41],
       [ 12,  12]], dtype=int64)
```

</td>
</tr>
<tr>
<td>K Neighbors Classifier</td>
<td>

```
knn = KNeighborsClassifier()
knn.fit(x_train_final, y_train)
```

KNeighborsClassifier

KNeighborsClassifier()

```
ypred_knn = knn.predict(x_test_final)
```

</td>
<td>

K Neighbors Classifier

```
knn_param_grid = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
knn_s = KNeighborsClassifier()
```

KNeighborsClassifier

KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')

</td>
</tr>
</table>

```
knn_acc = accuracy_score(ypred_knn, y_test)
knn_acc
```

```
0.6457142857142857
```

```
print(classification_report(y_test, ypred_knn))
```

```
              precision    recall  f1-score   support

           1       0.72      0.80      0.76       122
           2       0.38      0.28      0.33        53

    accuracy                           0.65       175
   macro avg       0.55      0.54      0.54       175
weighted avg       0.62      0.65      0.63       175
```

```
knn_cm = confusion_matrix(ypred_knn,y_test)
knn_cm
```

```
array([[98, 38],
       [24, 15]], dtype=int64)
```

```
ypred_knn = knn.predict(x_test_final)
```

```
knn_acc = accuracy_score(ypred_knn, y_test)
knn_acc
```

```
0.6628571428571428
```

```
print(classification_report(y_test, ypred_knn))
```

```
              precision    recall  f1-score   support

           1       0.74      0.80      0.77       122
           2       0.43      0.36      0.39        53

    accuracy                           0.66       175
   macro avg       0.59      0.58      0.58       175
weighted avg       0.65      0.66      0.65       175
```

```
knn_cm = confusion_matrix(ypred_knn,y_test)
knn_cm
```

```
array([[97, 34],
       [25, 19]], dtype=int64)
```

Final Model Selection Justification (2 Marks):

Final Model	Reasoning
Logistic Regression	Logistic Regression offers a balance of high interpretability, slightly better generalization performance, and computational efficiency, making it the best choice for this dataset given the similar accuracy and F1-scores among the models.