Model Optimization and Tuning Phase Template

Date	12 March 2024
Team ID	SWTID1720089323
Project Title	Ecommerce Shipping Prediction 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):

<table>
<tr><th>Model</th><th>Tuned Hyperparameters</th><th>Optimal Values</th></tr>
<tr><td>logistic regression</td><td><pre><code>lg = LogisticRegression(random_state=1000)
lg_param_grid = {
    'C': [0.01, 0.1, 1, 10, 100], #regularization strength
    'max_iter': [20, 100, 200], #iterations
    'random_state':[200,1000]
}
lg_cv = GridSearchCV(lg, lg_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
lg_cv.fit(x_train, y_train)</code></pre></td><td><pre><code>lg = LogisticRegression(C=0.1,max_iter=100,random_state=1000)
lg.fit(x_train,y_train)
print('Train Score:',lg.score(x_train,y_train))
print('Test Score:',lg.score(x_test,y_test))
y_pred = lg.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.6449180327868852
Test Score: 0.6290909090909091
Test Score(using predicted data): 62.909090909090914</code></pre></td></tr>
<tr><td>logistic regression CV</td><td><pre><code>lcv = LogisticRegressionCV(random_state=1000)
lcv_param_grid = {
    'Cs': [10, 15, 20], #regularization parameters
    'max_iter': [100, 200, 300]
}
lcv_cv = GridSearchCV(lcv, lcv_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
lcv_cv.fit(x_train, y_train)</code></pre></td><td><pre><code>lcv = LogisticRegressionCV(Cs= 15, max_iter= 100,random_state=1000)
lcv.fit(x_train,y_train)
print('Train Score:',lcv.score(x_train,y_train))
print('Test Score:',lcv.score(x_test,y_test))
y_pred = lcv.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.6474316939890711
Test Score: 0.6263636363636363
Test Score(using predicted data): 62.63636363636363</code></pre></td></tr>
</table>

<table>
<tr>
<td>XGBoost</td>
<td><pre>xgb = XGBClassifier(random_state=1000)
xgb_param_grid = {
    'min_child_weight': [1, 5, 10],
    'gamma': [0.5, 1, 5,10],
    'learning_rate':[0.1,0.9,1],
    'n_estimators': [100, 200, 300]
}
xgb_cv = GridSearchCV(xgb, xgb_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
xgb_cv.fit(x_train, y_train)</pre></td>
<td><pre>xgb = XGBClassifier(gamma= 10,learning_rate=1,random_state=1000,min_child_weight= 5,n_estimators= 100)
xgb.fit(x_train,y_train)
print('Train Score:',xgb.score(x_train,y_train))
print('Test Score:',xgb.score(x_test,y_test))
y_pred = xgb.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.7369398907103825
Test Score: 0.6809090909090909
Test Score(using predicted data): 68.0909090909091</pre></td>
</tr>
<tr>
<td>Ridge classifier</td>
<td><pre>rg = RidgeClassifier(random_state=1000)
rg_param_grid = {
    'alpha': [0.1, 1.0, 10.0, 100.0], #regularization strength
    'max_iter': [100, 200, 300]
}
rg_cv = GridSearchCV(rg, rg_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
rg_cv.fit(x_train, y_train)</pre></td>
<td><pre>rg = RidgeClassifier(random_state=1000,alpha=0.1,max_iter=100)
rg.fit(x_train,y_train)
print('Train Score:',rg.score(x_train,y_train))
print('Test Score:',rg.score(x_test,y_test))
y_pred = rg.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.6468852459016393
Test Score: 0.6284848484848485
Test Score(using predicted data): 62.84848484848485</pre></td>
</tr>
<tr>
<td>KNN</td>
<td><pre>knn = KNeighborsClassifier()
knn_param_grid = {
    'n_neighbors': [14,20,30],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree']
}
knn_cv = GridSearchCV(knn, knn_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
knn_cv.fit(x_train, y_train)</pre></td>
<td><pre>knn = KNeighborsClassifier(weights='distance',n_neighbors=14,algorithm='auto')
knn.fit(x_train,y_train)
print('Train Score:',knn.score(x_train,y_train))
print('Test Score:',knn.score(x_test,y_test))
y_pred = knn.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 1.0
Test Score: 0.6412121212121212
Test Score(using predicted data): 64.12121212121212</pre></td>
</tr>
<tr>
<td>Random
Forest</td>
<td><pre>rf = RandomForestClassifier(random_state=1000)
rf_param_grid = {
    'n_estimators': [50,100,150,200], #no of trees
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10,15, 20] ,
    'max_features': ['sqrt', 'log2'] ,
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

rf_cv = GridSearchCV(rf, rf_param_grid, cv=cv, scoring='accuracy', n_jobs=-1, verbose=3)
rf_cv.fit(x_train, y_train)</pre></td>
<td><pre>rf = RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=5,max_features='sqrt', n_estimators=2
rf.fit(x_train,y_train)
print('Train Score:',rf.score(x_train,y_train))
print('Test Score:',rf.score(x_test,y_test))
y_pred = rf.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.7665573770491804
Test Score: 0.6842424242424242
Test Score(using predicted data): 68.42424242424242</pre></td>
</tr>
<tr>
<td>Support
Vector
Classifier</td>
<td><pre>svc = svm.SVC(random_state=1000)

svc_param_grid = {
    'kernel': ['rbf','linear'],#considering poly requires higher computation power and requires more time
    'C': [1,3,10],
    'gamma': [0.1,5,10]
}

svc_grid_search = GridSearchCV(svc, param_grid=svc_param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=3)
svc_grid_search.fit(x_train, y_train)</pre></td>
<td><pre>svc = svm.SVC(random_state=1000,kernel='rbf',C= 10, gamma= 10)
svc.fit(x_train,y_train)
print('Train Score:',svc.score(x_train,y_train))
print('Test Score:',svc.score(x_test,y_test))
y_pred = svc.predict(x_test)#using predicted values
print('Test Score(using predicted data):',accuracy_score(y_test, y_pred) * 100)

Train Score: 0.9899453551912568
Test Score: 0.6209090909090909
Test Score(using predicted data): 62.090909090909086</pre></td>
</tr>
</table>

Performance Metrics Comparison Report (2 Marks):

Model	Baseline Metric	Optimized Metric
logistic regression	Classification Report: precision recall f1-score support 0 0.53 0.66 0.59 1312 1 0.73 0.61 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.64 0.63 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[870 442] [781 1207]]	Classification Report: precision recall f1-score support 0 0.53 0.67 0.59 1312 1 0.73 0.61 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.64 0.63 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[873 439] [785 1203]]
logistic regression CV	Classification Report: precision recall f1-score support 0 0.52 0.67 0.59 1312 1 0.73 0.59 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.63 0.62 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[884 428] [806 1182]]	Classification Report: precision recall f1-score support 0 0.52 0.67 0.59 1312 1 0.73 0.59 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.63 0.62 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[885 427] [806 1182]]
XGBoost	Classification Report: precision recall f1-score support 0 0.56 0.70 0.62 1312 1 0.76 0.64 0.70 1988 accuracy 0.66 3300 macro avg 0.66 0.67 0.66 3300 weighted avg 0.68 0.66 0.67 3300 Confusion Matrix: [[916 396] [718 1270]]	Classification Report: precision recall f1-score support 0 0.56 0.91 0.69 1312 1 0.90 0.53 0.67 1988 accuracy 0.68 3300 macro avg 0.73 0.72 0.68 3300 weighted avg 0.76 0.68 0.68 3300 Confusion Matrix: [[1190 122] [931 1057]]
Ridge classifier	Classification Report: precision recall f1-score support 0 0.53 0.67 0.59 1312 1 0.73 0.60 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.63 0.62 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[874 438] [789 1199]]	Classification Report: precision recall f1-score support 0 0.53 0.67 0.59 1312 1 0.73 0.60 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.64 0.62 3300 weighted avg 0.65 0.63 0.63 3300 Confusion Matrix: [[875 437] [789 1199]]

KNN	Classification Report: precision recall f1-score support 0 0.53 0.69 0.60 1312 1 0.74 0.59 0.66 1988 accuracy 0.63 3300 macro avg 0.63 0.64 0.63 3300 weighted avg 0.66 0.63 0.63 3300 Confusion Matrix: [[905 407] [812 1176]]	Classification Report: precision recall f1-score support 0 0.54 0.73 0.62 1312 1 0.77 0.58 0.66 1988 accuracy 0.64 3300 macro avg 0.65 0.66 0.64 3300 weighted avg 0.67 0.64 0.64 3300 Confusion Matrix: [[957 355] [829 1159]]
Random Forest	Classification Report: precision recall f1-score support 0 0.57 0.77 0.65 1312 1 0.80 0.61 0.69 1988 accuracy 0.67 3300 macro avg 0.68 0.69 0.67 3300 weighted avg 0.71 0.67 0.68 3300 Confusion Matrix: [[1009 303] [774 1214]]	Classification Report: precision recall f1-score support 0 0.56 0.94 0.70 1312 1 0.93 0.51 0.66 1988 accuracy 0.68 3300 macro avg 0.75 0.73 0.68 3300 weighted avg 0.78 0.68 0.68 3300 Confusion Matrix: [[1235 77] [965 1023]]
Support Vector Classifier	Classification Report: precision recall f1-score support 0 0.54 0.87 0.66 1312 1 0.85 0.51 0.64 1988 accuracy 0.65 3300 macro avg 0.70 0.69 0.65 3300 weighted avg 0.73 0.65 0.65 3300 Confusion Matrix: [[1139 173] [977 1011]]	Classification Report: precision recall f1-score support 0 0.52 0.50 0.51 1312 1 0.68 0.70 0.69 1988 accuracy 0.62 3300 macro avg 0.60 0.60 0.60 3300 weighted avg 0.62 0.62 0.62 3300 Confusion Matrix: [[655 657] [594 1394]]

Final Model Selection Justification (2 Marks):

Final Model	**Reasoning**
Random Forest	The Random Forest model was chosen as the final optimized model due to its superior performance metrics. It achieved the highest accuracy of 68.42%, demonstrating its effectiveness in making accurate predictions. Additionally, it exhibited a high precision score of 93.00%, indicating its reliability in correctly identifying true positives. Random Forest's ensemble approach helps in minimizing overfitting and improving

	generalization to new data. These characteristics align well with the project's objectives of enhancing delivery time predictions, making Random Forest the most suitable choice.