

## Model Development Phase Template

Date	July 2024
Team ID	Team-739789
Project Title	Auto Insurance Fraud Detection Using Machine Learning
Maximum Marks	10 Marks

### Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include a summary and training and validation performance metrics for multiple models, presented through respective screenshots.

### Initial Model Training Code (5 marks):

Paste the screenshot of the model training code

### Model Validation and Evaluation Report (5 marks):

Model	Summary	Training and Validation Performance Metrics
Model 1	Logistic regression model typically includes accuracy, precision, recall, F1 score to evaluate its predictive performance and generalization capability.	<pre>[ ] from sklearn.linear_model import LogisticRegression     from sklearn.metrics import accuracy_score      # ... (your existing code)      def comparison(x_test,y_test):         # Create and fit a logistic regression model         lrg = LogisticRegression()         lrg.fit(X_train, y_train) # Assuming X_train and y_train are defined          # Now you can calculate predictions         lrg_pred = lrg.predict(x_test)          print("logistic Regression:",100*accuracy_score(y_test,lrg_pred))         # ... (rest of your comparison function)      [ ] comparison(X_test, y_test) # Change 'x_test' to 'X_test' if it's defined that way</pre>

Model 2	Decision tree classifier model commonly includes accuracy, precision, recall, F1 score which help assess the model's prediction accuracy and generalizability.	<pre> from sklearn.tree import DecisionTreeClassifier from sklearn.metrics import accuracy_score  dtc=DecisionTreeClassifier()  # Replace Ellipsis with your actual training data X_train = [[1, 2], [3, 4], [5, 6]] # Example data, replace with your own y_train = [0, 1, 0] # Example labels, replace with your own  # Define X_test - replace with your actual test data X_test = [[7,8],[9,10],[11,12]] # Example data, replace with your own # Define y_test - replace with your actual test labels y_test = [1, 0, 1] # Example data, replace with your own  dtc.fit(X_train,y_train) y_pred=dtc.predict(X_test) dtc_train_acc=accuracy_score(y_train,dtc.predict(X_train)) dtc_test_acc=accuracy_score(y_test,y_pred) # You will need to define y_test as well print("Decision Tree metrics:") print("Train Accuracy:", dtc_train_acc) print("Test Accuracy:", dtc_test_acc) </pre> <p>Decision Tree metrics: Train Accuracy: 1.0 Test Accuracy: 0.3333333333333333</p>
Model 3	Random forest classifier model often encompasses accuracy, precision, recall, F1 score to measure its prediction quality and robustness.	<pre> from sklearn.ensemble import RandomForestClassifier import pandas as pd import numpy as np from sklearn.metrics import accuracy_score # Import accuracy_score  # Define X_train and y_train here with your actual data X_train = [[1, 2], [3, 4], [5, 6]] # Example data, replace with your own y_train = [0, 1, 0] # Example labels, replace with your own  # Define X_test - replace with your actual test data X_test = [[7,8],[9,10],[11,12]] # Example data, replace with your own # Define y_test - replace with your actual test labels y_test = [1, 0, 1] # Example data, replace with your own  # Initialize and fit the RandomForestClassifier (replace ... with your code) rfc = RandomForestClassifier() rfc.fit(X_train, y_train) y_pred = rfc.predict(X_test)  rfc_train_acc=100*accuracy_score(y_train,rfc.predict(X_train)) # Now accuracy_score is available rfc_test_acc=100*accuracy_score(y_test,y_pred) # ... (rest of your code) print("Random Forest metrics:") print("Train Accuracy:", rfc_train_acc) print("Test Accuracy:", rfc_test_acc) </pre> <p>Random Forest metrics: Train Accuracy: 100.0 Test Accuracy: 33.33333333333333</p>
Model 4	K-nearest neighbors' classifier model typically includes accuracy, precision, recall, F1 score to evaluate its prediction performance and generalization ability.	<pre> [1] from sklearn.neighbors import KNeighborsClassifier from sklearn.metrics import confusion_matrix, classification_report import numpy as np # Import numpy for array manipulation  # Assuming you have a larger x_train and corresponding y_train defined elsewhere, # use a subset of it that matches the size of your y_train in this example x_train_subset = x_train[:3] # Select the first 3 samples from your larger x_train  # Now x_train_subset and y_train have the same number of samples if x_train_subset.shape[0] &gt; 10 and y_train.shape[0] &gt; 0:     knn = KNeighborsClassifier(n_neighbors=30)     knn.fit(x_train_subset, y_train) # Fit the model with the subset     y_pred = knn.predict(x_test)     print(confusion_matrix(y_test, y_pred))     print(classification_report(y_test,y_pred))  # If you don't have a larger x_train, you need to define it here with 800 samples # to match the size mentioned in the error message. </pre>
Model 5	Naive Bayes classifier model typically includes accuracy, precision, recall, F1 score to evaluate its prediction performance and generalization.	<pre> from sklearn.naive_bayes import CategoricalNB,GaussianNB import numpy as np # Import numpy  # Define X_train - replace with your actual training data X_train = np.array([[1, 2], [3, 4], [5, 6]]) # Convert X_train to a Numpy array  # Check if X_train has data before fitting the model if X_train.shape[0] &gt; 0: # Use 'X_train' instead of 'x_train'     gnb=GaussianNB() # Reshape y_train to be a 1D array if it's not empty if len(y_train) &gt; 0: # Check if y_train has data using len() for lists     y_train_resaped = np.array(y_train.ravel()) # Reshape y_train to 1D using numpy model_2=gnb.fit(X_train,y_train_resaped) # Use X_train # Assuming X_test and y_test are defined elsewhere as Numpy arrays predict_log=model_2.predict(X_test) # Use X_test (uppercase) print("training accuracy",100*accuracy_score(model_2.predict(X_train),(y_train_resaped))) # Use X_train print("testing accuracy",100*accuracy_score(y_test,predict_log)) </pre>

<p>Model 6</p>	<p>Confusion matrix is used to evaluate the model's performance by showing the actual versus predicted classifications.</p>	<pre>def comparison(x_test, y_test):     # Create and fit a logistic regression model     lrg = LogisticRegression()     lrg.fit(X_train, y_train) # Assuming X_train and y_train are defined      # Now you can calculate predictions     lrg_pred = lrg.predict(x_test)      print("logistic Regression:", 100*accuracy_score(y_test, lrg_pred))     # ... (rest of your comparison function)      return lrg_pred # Return the lrg_pred variable  # Call the comparison function and store the result lrg_pred = comparison(X_test, y_test)  # Now you can print the classification report print(confusion_matrix(y_test, y_pred)) print(classification_report(y_test, lrg_pred))</pre> <p>Show hidden output</p> <p>[ ] Start coding or generate with AI.</p> <pre>print("confusion matrix\n", confusion_matrix(y_test, y_pred), "\n") print("classification_report\n", classification_report(y_test, y_pred))</pre> <p>Show hidden output</p> <p>Start coding or generate with AI.</p> <pre>[ ] print(confusion_matrix(y_test, y_pred)) print(classification_report(y_test, y_pred))</pre> <pre>[[1 0]  [2 0]]</pre> <p>precision recall f1-score support</p>
----------------	---	--