Research Project: A Comparison of Logistic Regression and Random Forests on Predicting Default of Credit Cards

Code Print-Out - 1

```
% Loading the raw data
data = readtable('/Users/aqsa/Desktop/ML Project/first_ML.csv',
'VariableNamingRule', 'preserve');
% Splitting into features (X) and target variable (Y)
X = data{:, 2:end-1}; % Excluding ID and DEFAULT columns
Y = data{:, end};     % DEFAULT column is the target
% Split into training and test sets (70:30 split)
cv = cvpartition(Y, 'Holdout', 0.3);
X_train = X(training(cv), :);
Y_train = Y(training(cv));
X_test = X(test(cv), :);
Y_test = Y(test(cv));
% Standardizing the training and test data
[X_train, mu, sigma] = zscore(X_train);
X_test = (X_test - mu) ./ sigma;
% Logistic Regression Model
lr_model = fitclinear(X_train, Y_train, 'Learner', 'logistic');
% Random Forest Model
rf_model = fitcensemble(X_train, Y_train, 'Method', 'Bag', 'NumLearningCycles',
100);
% Predictions for both models
pred_lr = predict(lr_model, X_test);
pred_rf = predict(rf_model, X_test);
% Confusion matrix for both models
confMat_lr = confusionmat(Y_test, pred_lr);
confMat_rf = confusionmat(Y_test, pred_rf);
% Calculating Precision, Recall, F1-score for Logistic Regression
TP_lr = confMat_lr(2, 2);
FP_lr = confMat_lr(1, 2);
FN_lr = confMat_lr(2, 1);
TN_lr = confMat_lr(1, 1);
precision_lr = TP_lr / (TP_lr + FP_lr);
recall_lr = TP_lr / (TP_lr + FN_lr);
f1_lr = 2 * (precision_lr * recall_lr) / (precision_lr + recall_lr);
% Calculating Precision, Recall, F1-score for Random Forest
TP_rf = confMat_rf(2, 2);
FP_rf = confMat_rf(1, 2);
FN_rf = confMat_rf(2, 1);
TN_rf = confMat_rf(1, 1);
%inspired by the github code for this from:
https://github.com/preethamam/MultiClassMetrics-ConfusionMatrix/blob/main/multi
class_metrics_common.m
% ^ this repository was linked out from the mathworks website for others to
% learn how to use the function
precision_rf = TP_rf / (TP_rf + FP_rf);
recall_rf = TP_rf / (TP_rf + FN_rf);
f1_rf = 2 * (precision_rf * recall_rf) / (precision_rf + recall_rf);
% Calculating AUC
```

```
[~, score_lr] = predict(lr_model, X_test);
[~, score_rf] = predict(rf_model, X_test);
[~, ~, ~, AUC_lr] = perfcurve(Y_test, score_lr(:,2), 1);
[~, ~, ~, AUC_rf] = perfcurve(Y_test, score_rf(:,2), 1);
%reference:
https://stackoverflow.com/questions/23696609/what-are-the-matlab-perfcurve-roc-curve-parameters
%reference: https://uk.mathworks.com/help/stats/perfcurve.html?#bunsogv-scores
% Calculating Log Loss (Cross-Entropy Loss)
% Add small epsilon to avoid log(0) or log(1) for log loss calculation
epsilon = 1e-15;
score_rf = max(min(score_rf, 1 - epsilon), epsilon);  % Clipping the values between epsilon and 1-epsilon
log_loss_lr = logloss(Y_test, score_lr(:,2));
log_loss_rf = logloss(Y_test, score_rf(:,2));
% I kept getting the 'loss' matlab function as an option when i googled
% things but stumbled upon this github that used the 'logloss' function to
% clip things for a binary classification
% (https://github.com/benhamner/Metrics/blob/master/MATLAB/metrics/logLoss.m).
% With the help of GenAI i was able to use this function correction to
% ensure i did not make any mistakes
% Display Test Scores for both models
fprintf('Logistic Regression Test Scores:\n');
fprintf('Accuracy: %.2f\n', sum(pred_lr == Y_test) / length(Y_test));
fprintf('Recall: %.2f\n', recall_lr);
fprintf('Precision: %.2f\n', precision_lr);
fprintf('F1-score: %.2f\n', f1_lr);
fprintf('AUC: %.2f\n', AUC_lr);
fprintf('Log Loss: %.2f\n', log_loss_lr);
fprintf('\nRandom Forest Test Scores:\n');
fprintf('Accuracy: %.2f\n', sum(pred_rf == Y_test) / length(Y_test));
fprintf('Recall: %.2f\n', recall_rf);
fprintf('Precision: %.2f\n', precision_rf);
fprintf('F1-score: %.2f\n', f1_rf);
fprintf('AUC: %.2f\n', AUC_rf);
fprintf('Log Loss: %.2f\n', log_loss_rf);
% ROC Curve Plotting for Logistic Regression
figure;
[X_lr, Y_lr, ~, AUC_lr] = perfcurve(Y_test, score_lr(:,2), 1); % Getting FPR and TPR for Logistic Regression
plot(X_lr, Y_lr, 'c', 'DisplayName', sprintf('Logistic Regression (AUC = %.2f)', AUC_lr));
hold on;
% ROC Curve Plotting for Random Forest
[X_rf, Y_rf, ~, AUC_rf] = perfcurve(Y_test, score_rf(:,2), 1); % Getting FPR and TPR for Random Forest
plot(X_rf, Y_rf, 'm', 'DisplayName', sprintf('Random Forest (AUC = %.2f)', AUC_rf));
xlabel('False Positive Rate');
```

```
ylabel('True Positive Rate');
title('ROC Curves');
legend show;
% Plot Confusion Matrix for both models
figure;
subplot(1, 2, 1);
heatmap(confMat_lr, 'Title', 'Confusion Matrix (Logistic Regression)',
'XLabel', 'Predicted', 'YLabel', 'Actual');
subplot(1, 2, 2);
heatmap(confMat_rf, 'Title', 'Confusion Matrix (Random Forest)', 'XLabel',
'Predicted', 'YLabel', 'Actual');
% Precision vs Recall Curve for both models
figure;
plot([0, 1], [recall_lr, precision_lr], 'r', 'DisplayName', 'Logistic
Regression');
hold on;
plot([0, 1], [recall_rf, precision_rf], 'b', 'DisplayName', 'Random Forest');
xlabel('Recall');
ylabel('Precision');
title('Precision vs Recall');
legend show;
% Metrics for Logistic Regression and Random Forest
metrics = {'Accuracy', 'Recall', 'Precision', 'F1-score', 'AUC', 'Log Loss'};
% List of metrics
values_lr = [sum(pred_lr == Y_test) / length(Y_test), recall_lr, precision_lr,
f1_lr, AUC_lr, log_loss_lr];  % Logistic Regression values
values_rf = [sum(pred_rf == Y_test) / length(Y_test), recall_rf, precision_rf,
f1_rf, AUC_rf, log_loss_rf];  % Random Forest values
% Grouped bar chart to compare metrics between Logistic Regression and Random
Forest
figure;
% Creating a grouped bar chart
bar_data = [values_lr; values_rf]';  % Arrange data so that each row
corresponds to a model and each column to a metric
b = bar(bar_data, 'grouped');  % Grouped bar chart
% Customizing the  bar colors so that there is one color for each metric
b(1).FaceColor = '#D8BFD8';  % blue for Logistic Regression
b(2).FaceColor = '#ADD8E6';  % pink for Random Forest
% Setting the x-axis labels (metrics)
set(gca, 'XTickLabel', metrics);
% Labeling the axes and the title
ylabel('Metric Value');
xlabel('Metrics');
title('Comparison of Logistic Regression and Random Forest across Different
Metrics');
% Displaying a legend
legend({'Logistic Regression', 'Random Forest'}, 'Location', 'NorthEast');
% Adding values on top of the bars for better visibility
for i = 1:length(metrics)
```

```
   text(i-0.15, bar_data(i, 1) + 0.02, sprintf('%.2f', bar_data(i, 1)),
'FontSize', 12);  % Logistic Regression values
   text(i+0.15, bar_data(i, 2) + 0.02, sprintf('%.2f', bar_data(i, 2)),
'FontSize', 12);  % Random Forest values
end
%Defining Log Loss function with the clipped values from above
function ll = logloss(y_true, y_pred)
 ll = -mean(y_true .* log(y_pred) + (1 - y_true) .* log(1 - y_pred));
end
```

Research Project: A Comparison of Logistic Regression and Random Forests on Predicting Default of Credit Cards

Code Print-Out - 2

```
% Loading the raw data
data = readtable('/Users/aqsa/Desktop/ML Project/first_ML.csv',
'VariableNamingRule', 'preserve');
% Splitting into features (X) and target variable (Y)
X = data{:, 2:end-1}; % Excluding ID and DEFAULT columns
Y = data{:, end};     % DEFAULT column is the target
% Split into training and test sets (70:30 split)
cv = cvpartition(Y, 'Holdout', 0.3);  % 30% test set
X_train = X(training(cv), :);
Y_train = Y(training(cv));
X_test = X(test(cv), :);
Y_test = Y(test(cv));
% Standardize the training and test data (code inspired from:
% https://gist.github.com/jsouza/4d2a8a3ba47bc82075d9)
[X_train, mu, sigma] = zscore(X_train);
X_test = (X_test - mu) ./ sigma;
fprintf('Starting Logistic Regression Tuning...\n'); %had to look up this
functionality (https://uk.mathworks.com/help/matlab/ref/tic.html)
tic; % Starts the  timing
% Random Search for Logistic Regression
lambda_random = logspace(-4, 0, 10); % Random search over regularization
solver_random = {'lbfgs', 'sgd', 'sparsa'}; % Random solvers for optimization
best_mcr_lr = inf;
for i = 1:length(lambda_random) %GENAI helped me debug my initial loop
   for j = 1:length(solver_random)
       % K-fold cross-validation setup
       cv_lr = cvpartition(Y_train, 'KFold', 5);
       mcr_fold = zeros(cv_lr.NumTestSets, 1);
       for k = 1:cv_lr.NumTestSets
           % Train-validation split
           X_train_fold = X_train(training(cv_lr, k), :);
           Y_train_fold = Y_train(training(cv_lr, k));
           X_val_fold = X_train(test(cv_lr, k), :);
           Y_val_fold = Y_train(test(cv_lr, k));
           % Train logistic regression
           model_lr = fitclinear(X_train_fold, Y_train_fold, 'Learner',
'logistic', ...
               'Lambda', lambda_random(i), 'Solver', solver_random{j});
           % Predict and calculate misclassification rate
           pred_val = predict(model_lr, X_val_fold);
           mcr_fold(k) = mean(pred_val ~= Y_val_fold);
       end
       % Evaluate the average misclassification rate
       avg_mcr = mean(mcr_fold);
       if avg_mcr < best_mcr_lr
           best_mcr_lr = avg_mcr;
           best_lambda_lr = lambda_random(i);
           best_solver_lr = solver_random{j};
```

```
        end
    end
end
% Train final logistic regression model with best hyperparameters
final_lr_model = fitclinear(X_train, Y_train, 'Learner', 'logistic', ...
   'Lambda', best_lambda_lr, 'Solver', best_solver_lr);
time_lr = toc; % End timing
fprintf('Best Logistic Regression - Lambda: %.4f, Solver: %s, MCR: %.4f\n', ...
   best_lambda_lr, best_solver_lr, best_mcr_lr);
fprintf('Logistic Regression Computational Time: %.2f seconds\n', time_lr);
%% Random Forest: Cross-Validation and Hyperparameter Tuning
fprintf('Starting Random Forest Tuning...\n');
tic; % Start timing
% Random Search: Define broad hyperparameter space
num_trees_random = [50, 100, 150];       % Number of trees
min_leaf_sizes_random = [1, 5, 10];      % Minimum leaf size (applies to trees
via template)
best_auc_rf = -inf;                      % Initialize best AUC
% Random Search
%GENAI helped me debug my initial loop
for i = 1:length(num_trees_random)
   for j = 1:length(min_leaf_sizes_random)
       % Define tree template with current min leaf size
       tree_template = templateTree('MinLeafSize', min_leaf_sizes_random(j));

       % K-fold cross-validation setup
       cv_rf = cvpartition(Y_train, 'KFold', 5);
       auc_fold = zeros(cv_rf.NumTestSets, 1); % Store AUC for each fold
       for l = 1:cv_rf.NumTestSets
           % Train-validation split
           X_train_fold = X_train(training(cv_rf, l), :);
           Y_train_fold = Y_train(training(cv_rf, l));
           X_val_fold = X_train(test(cv_rf, l), :);
           Y_val_fold = Y_train(test(cv_rf, l));
           % Train Random Forest
           model_rf = fitcensemble(X_train_fold, Y_train_fold, 'Method', 'Bag',
...
               'NumLearningCycles', num_trees_random(i), 'Learners',
tree_template);
           % Predict probabilities and calculate AUC
           [~, scores] = predict(model_rf, X_val_fold);
           [~, ~, ~, auc] = perfcurve(Y_val_fold, scores(:, 2), 1);
           auc_fold(l) = auc;
       end
       % Average AUC across folds
       avg_auc = mean(auc_fold);
       % Update best parameters if AUC improves
       if avg_auc > best_auc_rf
           best_auc_rf = avg_auc;
```

```
            best_num_trees_rf = num_trees_random(i);
            best_min_leaf_size_rf = min_leaf_sizes_random(j);
        end
    end
end
% this is a more focused Grid Search around the best values found in my Random
Search
fprintf('Starting Random Forest Grid Search...\n');
num_trees_grid = best_num_trees_rf - 20:10:best_num_trees_rf + 20; % Refine
around best number of trees
min_leaf_sizes_grid = max(1, best_min_leaf_size_rf - 2):best_min_leaf_size_rf +
2; % Refine leaf sizes
best_auc_rf_grid = -inf;
%GENAI helped me debug my this loop  below
for i = 1:length(num_trees_grid)
    for j = 1:length(min_leaf_sizes_grid)
        % Defined the tree template with current min leaf size
        tree_template = templateTree('MinLeafSize', min_leaf_sizes_grid(j));
        %referenced this for the code above:
https://uk.mathworks.com/help/stats/templatetree.html
        % K-fold cross-validation setup
        cv_rf = cvpartition(Y_train, 'KFold', 5);
        auc_fold = zeros(cv_rf.NumTestSets, 1);
        for l = 1:cv_rf.NumTestSets
            % Train-validation split
            X_train_fold = X_train(training(cv_rf, l), :);
            Y_train_fold = Y_train(training(cv_rf, l));
            X_val_fold = X_train(test(cv_rf, l), :);
            Y_val_fold = Y_train(test(cv_rf, l));
            % Train Random Forest
            model_rf = fitcensemble(X_train_fold, Y_train_fold, 'Method', 'Bag',
...
                'NumLearningCycles', num_trees_grid(i), 'Learners',
tree_template);
            % Predict probabilities and calculate AUC
            [~, scores] = predict(model_rf, X_val_fold);
            [~, ~, ~, auc] = perfcurve(Y_val_fold, scores(:, 2), 1);
            auc_fold(l) = auc;
        end
        % Averaged AUC across folds
        avg_auc = mean(auc_fold);
        % Updated best parameters if AUC improves
        if avg_auc > best_auc_rf_grid
            best_auc_rf_grid = avg_auc;
            best_num_trees_rf_grid = num_trees_grid(i);
            best_min_leaf_size_rf_grid = min_leaf_sizes_grid(j);
        end
    end
end
```

```
% Train final Random Forest model using the best hyperparameters
final_tree_template = templateTree('MinLeafSize', best_min_leaf_size_rf_grid);
final_rf_model = fitcensemble(X_train, Y_train, 'Method', 'Bag', ...
   'NumLearningCycles', best_num_trees_rf_grid, 'Learners',
final_tree_template);
time_rf = toc; % End timing
% Display Results
fprintf('Best Random Forest (Grid Search) - NumTrees: %d, MinLeafSize: %d, AUC:
%.4f\n', ...
   best_num_trees_rf_grid, best_min_leaf_size_rf_grid, best_auc_rf_grid);
fprintf('Random Forest Computational Time: %.2f seconds\n', time_rf);
%% Final Model Evaluation on Test Set
% Random Forest
[pred_rf, score_rf] = predict(final_rf_model, X_test);
[~, ~, ~, AUC_rf] = perfcurve(Y_test, score_rf(:, 2), 1);
% Display Final Test Metrics
fprintf('\nFinal Random Forest AUC: %.4f\n', AUC_rf);
%% Final Model Evaluation on Test Set
% Logistic Regression
[pred_lr, score_lr] = predict(final_lr_model, X_test);
[~, ~, ~, AUC_lr] = perfcurve(Y_test, score_lr(:, 2), 1);
% Random Forest
[pred_rf, score_rf] = predict(final_rf_model, X_test);
[~, ~, ~, AUC_rf] = perfcurve(Y_test, score_rf(:, 2), 1);
% Display Final Test Metrics
fprintf('\nFinal Logistic Regression AUC: %.4f\n', AUC_lr);
fprintf('Final Random Forest AUC: %.4f\n', AUC_rf);
%% Confusion Matrix and Visualization
% Logistic Regression Confusion Matrix
cm_lr = confusionmat(Y_test, pred_lr);
figure;
confusionchart(cm_lr, unique(Y_test), 'Title', 'Logistic Regression Confusion
Matrix', ...
   'RowSummary', 'row-normalized', 'ColumnSummary', 'column-normalized');
% Random Forest Confusion Matrix
cm_rf = confusionmat(Y_test, pred_rf);
figure;
confusionchart(cm_rf, unique(Y_test), 'Title', 'Random Forest Confusion
Matrix', ...
   'RowSummary', 'row-normalized', 'ColumnSummary', 'column-normalized');
%% Summary
fprintf('\nConfusion matrices plotted for both models.\n');
```