ML LAB Week 4
Model Selection and Comparative Analysis

Name: Cheruku Manas Ram
SRN: PES2UG23CS147
Section: C
Course Name: Machine Learning
Date: 01/09/2025

I. Introduction

The project compares manual grid search with scikit-learn's built-in GridSearchCV for hyperparameter tuning and combines multiple models in an attempt to get a better model.

1. **Hyperparameter Tuning**: Finding the best combination of hyperparameters for different classification models (Decision Tree, kNearestNeighbours and Logistic Regression) using both a manual implementation of grid search and scikit-learn's GridSearchCV.
2. **Model Comparison**: Evaluating the performance of the individually tuned models and combining them using voting classifiers to see if it improves performance.
3. **Visualization**: Visualizing model performance using ROC curves and confusion matrices.

II. Dataset Description

1. **Number of Instances**: The training set has 1029 instances and the testing set has 441 instances, for a total of 1470 instances.
2. **Number of Features**: The dataset has 46 features.
3. **Target Variable**: The target variable is 'Attrition', which represents whether an employee left the company ('Yes' or 1) or not ('No' or 0).

III. Methodology

1. **Hyperparameter Tuning**: Machine learning models have parameters that are learned from the data. They also have hyperparameters, which are settings that are not learned from the data but are set before the training process begins (eg: the number of neighbors in kNN, the maximum depth of a decision tree). Hyperparameter tuning is the process of finding the best set of hyperparameters for a model on a specific dataset to achieve optimal performance.
2. **Grid Search**: Grid search is a method for hyperparameter tuning. We define a grid of hyperparameter values. The algorithm then trains and evaluates the model for every possible combination of hyperparameters in the grid. The combination that yields the best performance is considered the optimal set of hyperparameters.

3. **K-Fold Cross-Validation**: K-Fold Cross-Validation is used to evaluate a model's performance and see how well it generalises unseen data. The dataset is split into k equally sized folds. The model is then trained k times. In each iteration, one fold is used as the validation set, and the remaining k-1 folds are used as the training set. The performance metric is calculated for each iteration on the validation fold, and the average of these k scores is taken as the final performance estimate.

ML Pipeline Used:

1. **StandardScaler**: This step standardizes features by removing the mean and scaling to unit variance. This is important for many algorithms (like Logistic Regression and KNN) that are sensitive to the scale of the input features.
2. **SelectKBest**: This step performs feature selection. SelectKBest selects the top k features based on a scoring function. I used “f_classif” as a scoring function that computes the ANOVA F-value between the features and the target variable. The k in SelectKBest is a hyperparameter that is tuned during the grid search.
3. **Classifier**: This is the final step, which is the machine learning model itself (Decision Tree, KNN, or Logistic Regression). The hyperparameters of this classifier are also tuned during the grid search.

These steps are chained together using scikit-learn's Pipeline class, which ensures that the data is processed sequentially through these steps during training and prediction.

- **Manual Grid Search Implementation**:
    1. For each classifier (Decision Tree, KNN, Logistic Regression):
    2. The parameter grid for feature selection (feature_selection__k) is adjusted to ensure that the number of selected features does not exceed the total number of features in the dataset.
    3. All possible combinations of hyperparameters from the adjusted grid are generated.
    4. For each parameter combination, 5-Fold Cross-Validation is performed manually.
    5. In each fold of the cross-validation:
        - The data is split into training and validation sets.
        - A pipeline (Scaler -> Feature Selection -> Classifier) is created.
        - The hyperparameters of the pipeline are set to the current combination.
        - The pipeline is trained on the training fold.
        - Probabilities are taken for each validation fold.
        - The ROC AUC score is calculated for the fold.
    6. The mean ROC AUC score across all 5 folds is calculated for the parameter combination.
    7. The parameter combination that yields the highest mean ROC AUC score is identified as the best.
    8. A final pipeline is created with the best hyperparameters found and trained on the entire training dataset.
    9. This best-tuned pipeline is taken.
- **Scikit-learn Grid Search Implementation**:

1. For each classifier:
2. The parameter grid for feature selection is adjusted as in the manual implementation.
3. A pipeline (Scaler -> Feature Selection -> Classifier) is created.
4. An instance of GridSearchCV is created, providing the pipeline the adjusted parameter grid, the cross-validation strategy, and the scoring metric. n_jobs=-1 is used to parallelize the search across multiple CPU cores.
5. grid_search.fit() is called on the training data. GridSearchCV internally performs the K-Fold Cross-Validation for each parameter combination in the grid, evaluates the performance using the specified scoring metric, and identifies the best parameter combination.
6. The best estimator (pipeline trained with the best hyperparameters on the entire training data), cross-validation score, and the best parameters found by GridSearchCV are retrieved and stored.

Both implementations aim to find the best hyperparameters using cross-validation, but the scikit-learn version automates the process.

IV. Results and analysis

1. Performance Tables:

Manual Grid Search Performance:

Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Decision Tree	0.8231	0.3333	0.0986	0.1522	0.7107
KNN	0.8435	0.5833	0.0986	0.1687	0.6854
Logistic Regression	0.8798	0.7368	0.3944	0.5138	0.8177
Voting Classifier	**0.8549**	**0.8182**	**0.1268**	**0.2195**	**0.7890**

Built-in Grid Search Performance:

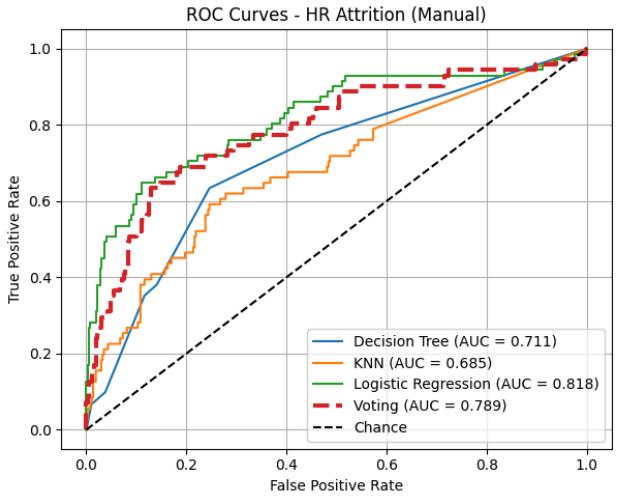
Model	Accuracy	Precision	Recall	F1-Score	ROC AUC
Decision Tree	0.8231	0.3333	0.0986	0.1522	0.7107
KNN	0.8435	0.5833	0.0986	0.1687	0.6854
Logistic Regression	0.8798	0.7368	0.3944	0.5138	0.8177
Voting Classifier	**0.8526**	**0.7143**	**0.1408**	**0.2353**	**0.7890**

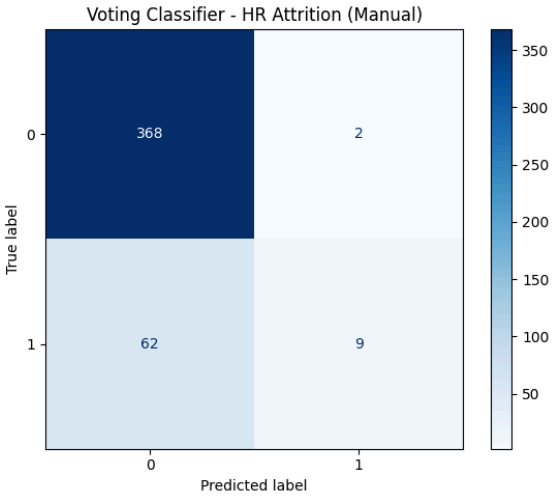
2. Compare Implementations:

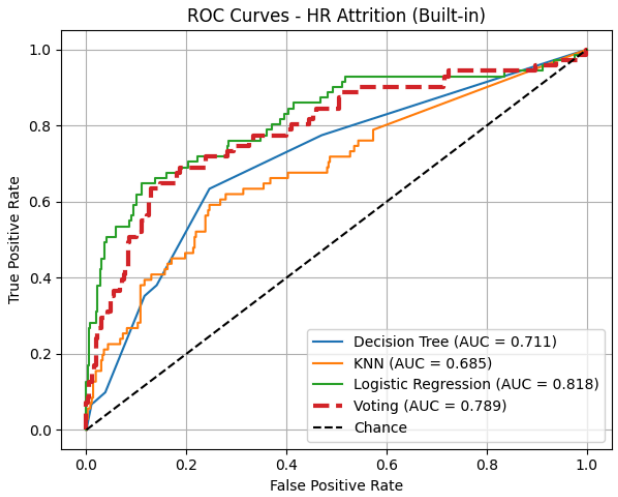
For the individual models, the performance metrics are identical between the manual and built-in grid search implementations. This indicates that both implementations successfully found the same best hyperparameters and resulted in the same trained models when evaluated on the test set.

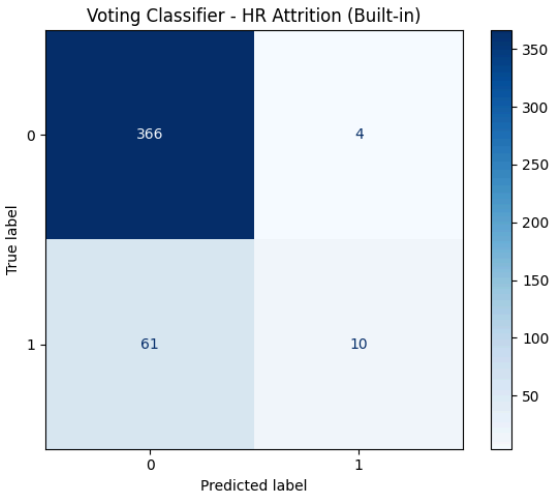
There are minor differences in the performance of the Voting Classifiers. The Manual Voting Classifier shows slightly higher Precision and Accuracy, while the Built-in Voting Classifier shows slightly higher Recall and F1-Score. The ROC AUC is identical for both. These minor differences in the voting classifier metrics is probably due to how it handles ties or small floating-point differences in probability predictions from the models.

3. Visualisations









Analysis of Plots:

- **ROC Curves**: The ROC curves visually represent the trade-off between the True Positive Rate and the False Positive Rate. A curve that is closer to the top-left corner indicates better performance. In our dataset, the Logistic Regression model (green

dashed line) shows the highest AUC (0.818), suggesting it has the best ability to distinguish between the two classes across different thresholds. The Decision Tree and KNN models have lower AUC values, indicating poorer performance compared to logistic regression. The Voting Classifier (red dashed line) performs better than Decision Tree and KNN, but slightly below the Logistic Regression model in terms of AUC. The manual and built-in plots are visually identical.

- **Confusion Matrices**: The confusion matrices show the number of correct and incorrect predictions made by the Voting Classifier on the test set.
    - The top-left cell (True Negatives) shows the number of employees who did not attrit and were correctly predicted as not attriting.
        - Manual - 368
        - Built in - 366
    - The top-right cell (False Positives) shows the number of employees who did not attrit but were incorrectly predicted as attriting.
        - Manual - 2
        - Built in - 4
    - The bottom-left cell (False Negatives) shows the number of employees who attrited but were incorrectly predicted as not attriting.
        - Manual - 62
        - Built in - 61
    - The bottom-right cell (True Positives) shows the number of employees who attrited and were correctly predicted as attriting.
        - Manual - 9
        - Built in - 10

The matrices show that both voting classifiers have a relatively high number of False Negatives, which means they struggle to correctly identify employees who actually attrit. This is shown in the relatively low Recall scores for the voting classifiers. The number of False Positives is quite low, leading to a high precision. The manual and built-in confusion matrices are very similar, with only slight differences in the counts.

4. Best Model

Based on the ROC AUC score on the test set, the **Logistic Regression** model performed best among the individual classifiers (AUC = 0.8177). The Voting Classifier achieved an AUC of 0.7890, which is lower than Logistic Regression.

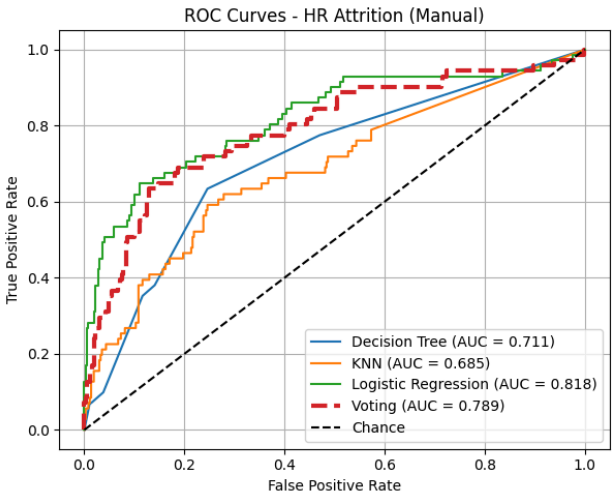
Hypothesis for why Logistic Regression performed best:

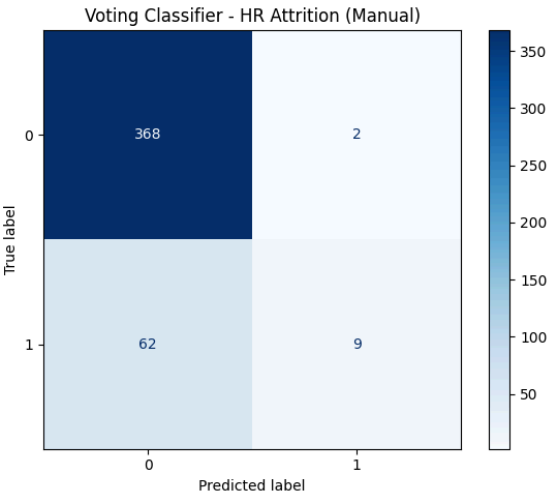
Logistic Regression is a linear model that works well when there is a relatively linear relationship between the features and the target variable. The HR Attrition dataset, might have many binary or categorical features, and Logistic Regression is often effective in modeling such data. Decision Trees can sometimes overfit, and KNN's performance is highly dependent on the distance metric and number of neighbors, and may struggle with high-dimensional data or irrelevant features, even with feature selection. Logistic Regression's ability to model the probability of attrition based on a linear combination of features seems to be a good fit for this dataset.

V. Screenshots

```
######################################################################
PROCESSING DATASET: HR ATTRITION
######################################################################
IBM HR Attrition dataset loaded and preprocessed successfully.
Training set shape: (1029, 46)
Testing set shape: (441, 46)
----------------------------

============================================================
RUNNING MANUAL GRID SEARCH FOR HR ATTRITION
============================================================
--- Manual Grid Search for Decision Tree ---
--------------------------------------------------------------------------------
Best parameters for Decision Tree: {'feature_selection__k': 5, 'classifier__max_depth': 3, 'classifier__min_samples_leaf': 4, 'classifier__min_samples_split': 2}
Best cross-validation AUC: 0.7176
--- Manual Grid Search for KNN ---
--------------------------------------------------------------------------------
Best parameters for KNN: {'feature_selection__k': 'all', 'classifier__n_neighbors': 9, 'classifier__p': 1, 'classifier__weights': 'distance'}
Best cross-validation AUC: 0.7201
--- Manual Grid Search for Logistic Regression ---
--------------------------------------------------------------------------------
Best parameters for Logistic Regression: {'feature_selection__k': 'all', 'classifier__penalty': 'l2', 'classifier__C': 0.1}
Best cross-validation AUC: 0.8328
```

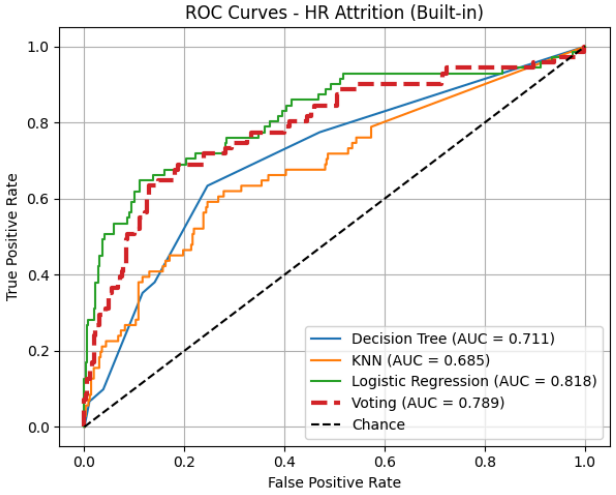
```
============================================================
EVALUATING MANUAL MODELS FOR HR ATTRITION
============================================================

--- Individual Model Performance ---

Decision Tree:
  Accuracy: 0.8231
  Precision: 0.3333
  Recall: 0.0986
  F1-Score: 0.1522
  ROC AUC: 0.7107

KNN:
  Accuracy: 0.8435
  Precision: 0.5833
  Recall: 0.0986
  F1-Score: 0.1687
  ROC AUC: 0.6854

Logistic Regression:
  Accuracy: 0.8798
  Precision: 0.7368
  Recall: 0.3944
  F1-Score: 0.5138
  ROC AUC: 0.8177

--- Manual Voting Classifier ---
Voting Classifier Performance:
  Accuracy: 0.8549, Precision: 0.8182
  Recall: 0.1268, F1: 0.2195, AUC: 0.7890
```

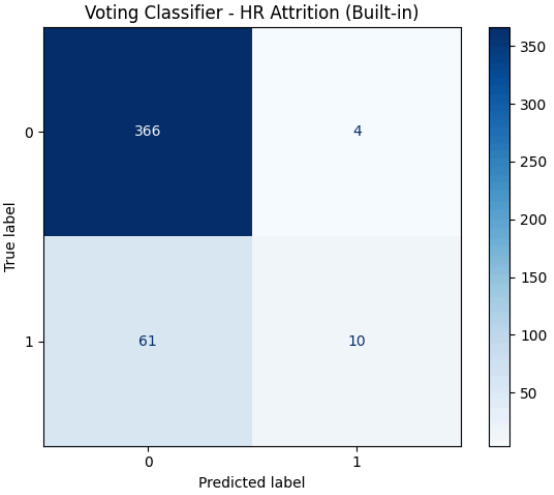





```
============================================================
RUNNING BUILT-IN GRID SEARCH FOR HR ATTRITION
============================================================

--- GridSearchCV for Decision Tree ---
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Best params for Decision Tree: {'classifier__max_depth': 3, 'classifier__min_samples_leaf': 4, 'classifier__min_samples_split': 2, 'feature_selection__k': 5}
Best CV score: 0.7176

--- GridSearchCV for KNN ---
Fitting 5 folds for each of 96 candidates, totalling 480 fits
Best params for KNN: {'classifier__n_neighbors': 9, 'classifier__p': 1, 'classifier__weights': 'distance', 'feature_selection__k': 'all'}
Best CV score: 0.7201

--- GridSearchCV for Logistic Regression ---
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Best params for Logistic Regression: {'classifier__C': 0.1, 'classifier__penalty': 'l2', 'feature_selection__k': 'all'}
Best CV score: 0.8328
```

```
============================================================
EVALUATING BUILT-IN MODELS FOR HR ATTRITION
============================================================

--- Individual Model Performance ---

Decision Tree:
  Accuracy: 0.8231
  Precision: 0.3333
  Recall: 0.0986
  F1-Score: 0.1522
  ROC AUC: 0.7107

KNN:
  Accuracy: 0.8435
  Precision: 0.5833
  Recall: 0.0986
  F1-Score: 0.1687
  ROC AUC: 0.6854

Logistic Regression:
  Accuracy: 0.8798
  Precision: 0.7368
  Recall: 0.3944
  F1-Score: 0.5138
  ROC AUC: 0.8177

--- Built-in Voting Classifier ---
Voting Classifier Performance:
  Accuracy: 0.8526, Precision: 0.7143
  Recall: 0.1408, F1: 0.2353, AUC: 0.7890
```





```
Completed processing for HR Attrition
================================================================================

================================================================================
ALL DATASETS PROCESSED!
================================================================================
```

VI. Conclusion

Key Findings:

- For the HR Attrition dataset, the **Logistic Regression model** performed best among the individual classifiers in terms of ROC AUC (0.8177).
- The Manual Grid Search and Built-in Grid Search implementations yielded identical best hyperparameters and individual model performance. Hence, manual implementation correctly replicated the logic of the built-in version for this dataset.
- There were only minor differences in the performance metrics of the Voting Classifiers between the manual and built-in approaches.
- The Voting Classifier's performance for HR Attrition (AUC 0.7890) was slightly lower than the best individual model (Logistic Regression). This highlights that simply combining models doesn't always guarantee better performance than the best individual component.

Takeaways:

1. Implementing grid search and cross-validation manually provides a deeper understanding of the underlying mechanics like how parameter combinations are explored, how data is split for validation, and how performance is evaluated iteratively.
2. However, Scikit-learn's GridSearchCV simplifies and accelerates the hyperparameter tuning process. It is concise, has built in functions and allows for parallel processing.
3. While the Voting Classifier didn't outperform logistic regression on HR Attrition, ensembling is a powerful technique that often improves performance by combining the strengths of different models and reducing variance.

While manual implementation is great for learning and understanding, using powerful libraries like scikit-learn is essential for practical, efficient machine learning workflows, especially when dealing with complex models and large parameter spaces. The lab effectively demonstrates this trade-off while also showing how to build and evaluate model ensembles.