ML LAB 4

Week 4: Model Selection and Comparative Analysis

Name: Darshan N	SRN: PES2UG24CS808	SEC: C

Part 1: Getting the Tools Ready

Before we do anything with machine learning, we need to bring in the right tools. Libraries like pandas and numpy help us handle data, while scikit-learn gives us all the machine learning functions we need. Think of this step as setting up your toolbox before starting a project—it makes sure everything you'll need later is already in place.

Part 2: Choosing Models and Their Settings

We'll be working with three simple but powerful models:

- Decision Tree
- k-Nearest Neighbors (kNN)
- Logistic Regression

Each of these models has settings (called hyperparameters) that control how they behave. For example, a decision tree has a maximum depth, and kNN has the number of neighbors. We write down a list of these settings, called a parameter grid, so we can test which combination works best.

Part 3: Getting the Data (HR Attrition Example)

Our dataset is about predicting whether employees will leave a company or stay. Here's what we do:

1. Load the file from the data/ folder.
2. Change the "Attrition" column into numbers (yes = 1, no = 0).

3. Turn text-based columns (like job role or department) into numerical values using one-hot encoding.
4. Split the data into training and testing sets, making sure both groups have a fair balance of “yes” and “no” cases.

Part 4: Doing Grid Search Manually

Instead of using built-in tools, we first try to search for the best settings ourselves.

- We create all possible hyperparameter combinations.
- Test them one by one using 5-fold cross-validation (splitting the data into 5 parts to test and train repeatedly).
- Measure performance using ROC AUC score.
- Finally, pick the best settings and train the model with them.

It’s like cooking—you try different amounts of salt, spices, and heat until you find the tastiest version.

Part 5: Using GridSearchCV (The Automatic Way)

Scikit-learn has a tool called GridSearchCV that does the above process automatically.

- We set up a pipeline that includes preprocessing (scaling numbers, selecting features) and the model.
- Tell it which parameters to try.
- It runs the same 5-fold cross-validation, but much faster and cleaner.
- At the end, it gives us the best model with the best parameters.

Part 6: Checking and Combining Models

After tuning, we need to see how good each model is. We check them using:

- Accuracy

- Precision & Recall
- F1-score
- ROC AUC

We also draw confusion matrices (to see where the model makes mistakes) and ROC curves (to see how well it separates classes).

Finally, we combine models into a voting classifier—basically, let all three models "vote" on the answer. This often improves performance since multiple models working together are usually smarter than one.

Part 7: One Complete Pipeline

Instead of doing all these steps separately, we create a single function, run_complete_pipeline(). This function does everything in order:

- Loads the data
- Runs manual grid search
- Runs automated grid search
- Evaluates results with metrics and visuals

This way, you just run one command, and the entire experiment is completed end-to-end.

Part 8: Running the Whole Lab

In the final step, we actually execute the pipeline on our dataset. This ensures that:

- Both manual and automated tuning are done.
- All metrics are calculated.
- Visual comparisons are generated.

At the end, we have a clear picture of how different models perform, how manual vs. automatic tuning compare, and whether combining models gives us better results.

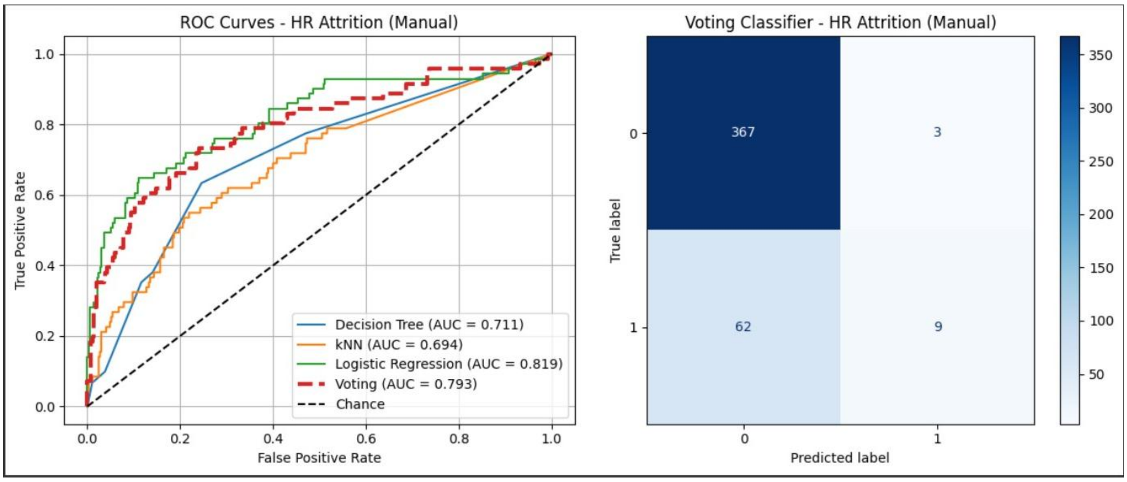
In short:
We set up the tools → pick models & parameters → prepare data → test parameters manually → test them automatically → evaluate results → combine models → run everything in one smooth pipeline.

Output Screenshots:

```
datasets = [
    (load_hr_attrition, "HR Attrition")
]

# Run for each dataset
for dataset_loader, dataset_name in datasets:
    try:
        run_complete_pipeline(dataset_loader, dataset_name)
    except Exception as e:
        print(f"Error processing {dataset_name}: {e}")
        continue

print("\n" + "="*80)
print("ALL DATASETS PROCESSED!")
print("="*80)
```

```
################################################################################
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
============================================================
```

```
Best cross-validation AUC: 0.8329

============================================================
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: 0.8413
  Precision: 0.5455
  Recall: 0.0845
  F1-Score: 0.1463
  ROC AUC: 0.6936

Logistic Regression:
  Accuracy: 0.8798
  Precision: 0.7368
  Recall: 0.3944
  F1-Score: 0.5138
  ROC AUC: 0.8187

--- Manual Voting Classifier ---
Voting Classifier Performance:
  Accuracy: 0.8526, Precision: 0.7500
  Recall: 0.1268, F1: 0.2169, AUC: 0.7933
```




```
================================================================
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: 0.8413
  Precision: 0.5455
  Recall: 0.0845
  F1-Score: 0.1463
  ROC AUC: 0.6936

Logistic Regression:
  Accuracy: 0.8798
  Precision: 0.7368
  Recall: 0.3944
  F1-Score: 0.5138
  ROC AUC: 0.8187
```