CS 486 Fall 2025 Homework 2 Cade Mendoza, jcmendoza2@dons.usfca.edu, September 26, 2025

Audit of Original Database

The dataset provided comes from the shared efforts of the J. Craig Venter Institute, Allen Institute for Brain Science, Department of Biomedical Informatics at University at Buffalo etc. The dataset originates from the biological study on e1 cluster classifications in human brain tissue. The ground truth for e1 clusters was established through independent biological research which identified specific gene expression patterns characteristic of "human middle temporal gyrus cortical layer 1 excitatory neurons". According to the research paper, e1 clusters are defined as neurons that "selectively express TESPA1, LINC00507 and SLC17A7 mRNAs and lack expression of KNCP1 mRNA". Taking this definition into account, we are provided a validated reference standard for evaluating machine learning classification performances.

Shown in Table 1, the original training database (.csv) contains 871 total samples with 608 gene expression features plus one class label column.

Table 1

Number of Samples	871	**Number of Features**	608

Table 2 shows the dataset which represents a binary classification problem with the following characteristics:

Table 2

Class Distribution		
Class	**Sample Count**	**Percentage**
Class 0	572	65.7%
Class 1	299	34.3%
Other Details		
Missing Values	0	
Data Types	All Numerical Features	

Class 0 (non-e1) contains 572 samples (65.7%) while Class 1 (e1) contains 299 samples (34.3%). All 608 features are numerical gene expression values (standardized nomenclature) and the data presents no missing values detected which eliminates the need for imputation and is formatted consistently across all features. The class labels are biologically validated through the independent research that established the e1 cluster phenotype and the ground truth is defined with the specific gene expression markers. The samples represent human cortical cell types from the middle temporal gyrus region, thus providing focused but limited brain region coverage but is appropriate for the specific biological research question being posed. In terms of class balance, there is no severe class imbalance as both classes exceed the 10% threshold making it a moderate imbalance that is manageable for Random Forest algorithms. The sample to feature ratio is 1.4:1

(871 samples % 608 features) which falls significantly below the recommended 10:1 ratio which presents the potential challenge for poor generalization to new data and overfitting. Aside from that, the gene expression data contains no personally identifiable information which ensures privacy protection. While the low sample-to-feature ratio presents challenges, this is typical in genomics datasets and Random Forest is suited for this scenario due to its built-in feature selection and resistance to overfitting through bootstrap sampling.

Creation of *training DB* and *verification DB*

Following assignment specifications, the original database was divided into training and verification datasets; one positive sample (Class 1) and one negative sample (Class 0) were randomly selected using pandas .sample() method with random_state=42 for reproducibility and were removed to create the verification database. Splitting the two sets, the training database holds 869 samples and the verification database holds 2 samples as shown in Table 3.

Table 3

Database	**Number of Samples**
Training Database	869
Verification Database	2

The original dataset was first separated by class labels to enable balanced sampling. With random selection, as mentioned above, one sample from each class was randomly selected using controlled seeding. The verification database was created by combining the two selected samples, while the training database consisted of all the remaining samples. *Refer to table 4.*

Table 4

Class Separation
class_0_samples = df[df['Label'] == 0] class_1_samples = df[df['Label'] == 1]
Random Selection
verify_0 = class_0_samples.sample(n=1, random_state=42) verify_1 = class_1_samples.sample(n=1, random_state=42) verify_db = pd.concat([verify_0, verify_1])

The training database has 869 samples, reduced from the original 871 and still possesses 608 gene expression values. Its purpose now is to be used for model training, hyperparameter optimization, OOB, and cross validation. The verification database contains 2 samples, one from each class, and possesses the same 608 gene expression values and will be used for runtime testing. *Refer to table 5.*

Table 5

	Training Database		**Verification Database**	
Class 0	571 samples	65.7%	1 sample	50%
Class 1	298 samples	34.3%	1 sample	50%

SW Tools

- Python 3.x with Jupyter Notebook
- Scikit-Learn for Random Forest implementation (ML algorithms and evaluation metrics)
- pandas for data manipulation
- numpy for numerical operations
- RandomForestClassifier, StratifiedKFold, and comprehensive metrics from sklearn
- Professor Petkovic's provided pseudo-code and hints documents
- Scikit-learn documentation
- Perplexity AI on brief guide setup
- Youtube: Getting Started with SciKit
    - How to use Random Forest in SciKit
    - OOB Score Explained in SciKit
    - Difference Between OOB and CV
    - Cross Validation Explained in SciKit
    - Computing Confusion Matrix, F1 scores, Precision and More in SciKit
- Anaconda Assistant (help on code syntax)
- OOB Errors for Random Forest in Scikit Learn GeeksforGeeks

Experimental Methods and Setup

I will be employing a grid search approach across a selected hyperparameter range. This parameter selection was guided by established best practices and guidance from Professor Petkovic. MTRY or max_features is calculated as follows:

Table 6

MTRY

max_features_range = [
  int(0.5 * sqrt_features),
  sqrt_features,
  int(2 * sqrt_features)
]

Calculated as [0.5×√608, √608, 2×√608] where 608 is the total number of features, this range explores conservative feature samples (12) to aggressive (48) selection. Another feature in the hyperparameter range is n_estimators or NTREE which represents the number of decision trees

in the Random Forest ensemble. I will be working with NTREE values of 500, 1000, and 2000. This range allows for a balance of computational efficiency with the Random Forest ensemble with higher values providing more stable predictions but also increasing computational cost.

Table 7

n_estimators (NTREE)	500	1000	2000
max_features (MTRY)	~12	~25	~49
Cutoff	0.5 (unable to be changed in scikit-learn)		
Accuracy Evaluation	Out-of-bounds scoring (OOB)	3-Fold Cross Validation (CV)	

The classification threshold (Cutoff) is fixed at 0.5. SciKit-learn implementation uses majority vote by default and the cutoff cannot be modified in SciKit-learn RandomForestClassifier. The Cutoff represents the decision boundary for the binary classification. In terms of the design approach, I will be incorporating dual validation to ensure robust performance estimation using Out-of-Bag (OOB) scoring and 3-Fold Stratified Cross Validation (CV). Out-of-Bag Scoring will utilize samples that are not selected during bootstrap sampling for each tree which will provide unbiased error estimation without needing separate validation data. This will be implemented through *oob_score=True* parameter in RandomForestClassifier. For 3-Fold Stratified Cross-Validation, it will be utilized to maintain class proportions across all folds and will be used to reduce variance in performance estimates if we were to compare it to simple random splits. The validation will be implemented using: *StratifiedKFold(n_splits=3, shuffle=True, random_state=42).*

The primary API's and Classes that I will be utilizing are RandomForestClassifier which is the core ensemble learning algorithm (possessing the parameters: n_estimators, max_features, oob_score, random_state, bootstrap), StratifiedKFold which is a cross-validation strategy preserving class ratios (possessing the parameters: n_splits=3, shuffle=True, random_state=42) and sklearn.metrics Module which possesses comprehensive evaluation functions (accuracy_score, precision_score, recall_score, f1_score etc.). All classification performance will be evaluated using standard binary classification metrics:

Accuracy	(TP + TN) / (TP + TN + FP + FN)
Precision	TP / (TP + FP)
Recall	TP / (TP + FN)
F1 Score	2 x (Precision x Recall) / (Precision + Recall)

In order to ensure consistent results across multiple executions, identical cross-validation fold assignments, and reproducible bootstrap sampling in RF, all random processes will utilize random_state=42.

The process will flow in this sequence:

1. Grid search across all hyperparameter combinations (3 x 3 = 9 configurations)
2. OOB Scoring during RF training for each configuration
3. 3-Fold Cross Validation for independent performance validation
4. Selection of optimal hyperparameters based on highest performance scores
5. Final model training using the optimal parameters on the full training dataset
6. Performance comparison between OOB and CV estimates

Actual Results of RF Training and Accuracy Estimates

The Random Forest training was implemented using a systematic grid search approach with the following key code components in Table 8:

Table 8

```
# Grid search implementation
for n_est in n_estimators_range:
    for max_feat in max_features_range:
        rf = RandomForestClassifier(
            n_estimators=n_est,
            max_features=max_feat,
            oob_score=True,
            random_state=42,
            bootstrap=True
        )
        rf.fit(X_train, y_train)
```

The training process utilized StratifiedKFold Cross-Validation to main class proportions across folds. A comprehensive evaluation of 9 parameter combinations was conducted, testing 3 n_estimator values (500, 1000, 2000) against 3 max_feature values (12, 24, 48). Both OOB Scoring and 3-Fold CV were employed to ensure a robust performance assessment. The table below reports the OOB Scoring results. The OOB accuracy scores for 9 RF configurations trained on 869 e1 cell classification samples (608 gene expression features) using the parameters: bootstrap=True and random_state=42 for reproducibility. The OOB evaluation during the training phase yielded the following performance matrix in Table 9.

Table 9

RF Training with OOB Scoring			
n_estimators	**max_features = 12**	**max_features = 24**	**max_features = 48**

500	0.99540	0.99540	0.99310
1000	0.99540	0.99540	0.99310
2000	0.99540	0.99540	0.99310

The best OOB parameters was collected simultaneously during the grid search as seen in Table 10:

Table 10

```
oob_score = rf.oob_score_
        print(f"n_est={n_est}, max_feat={max_feat}: OOB={oob_score:.5f}")
        if oob_score > best_OOB_score:
            best_OOB_score = oob_score
            best_OOB_params = {'n_estimators': n_est, 'max_features': max_feat}
            best_OOB_model = rf
```

It was found that the best OOB parameters were n_estimators = 500, max_features = 12, and OOB Score = 0.99540.

The 3-Fold CV accuracy scores used StratifiedKFold(n_splits=3, shuffle=True, random_state=42) on 869 training samples maintaining original class proportions (65.7% Class 0, 34.3% Class 1). Each fold contained around 290 samples and the stratification ensured balanced representation of e1 and non-e1 cell types across all folds. Performance was averaged across three independent train/test partitions to reduce variance in estimates as seen in the following charts. 3-Fold CV produced consistent results. *Refer to Table 11.*

Table 11

3-Fold Cross Validation			
n_estimators	**max_features = 12**	**max_features = 24**	**max_features = 48**
500	0.99540	0.99424	0.99195
1000	0.99540	0.99540	0.99195
2000	0.99540	0.99425	0.99195

The best CV parameters were collected simultaneously during the grid search. *Refer to table 12.*

Table 12

```
        avg_CV_score = np.mean(fold_scores)
        print(f"n_est={n_est}, max_feat={max_feat}: CV={avg_CV_score:.5f}")

        if avg_CV_score > best_CV_score:
                best_CV_score = avg_CV_score
                best_CV_params = {'n_estimators': n_est, 'max_features': max_feat}
                all_FOLD_CMS = fold_cms
                all_FOLD_Predicts = fold_predicts
                all_FOLD_True_labels = fold_true_labels
```

It was found that the best CV parameters were n_estimators = 500, max_features = 12, CV Score = 0.99540. With the best performing model configuration being n_estimators = 500, max_features = 12, Table 13 is a detailed individual fold-by-fold confusion matrix.

Table 13

Individual Fold Analysis for Optimal Parameters				
Fold	**Actual**	**Predicted 0**	**Predicted 1**	**Accuracy**
1	0	190	1	0.99655 (289/290 correct)
	1	0	99	
2	0	189	1	0.99655 (289/290 correct)
	1	0	100	
3	0	189	1	0.99312 (288/290 correct)
	1	1	98	

The aggregated confusion matrix across all three folds is provided in Table 14. Rows (Actual) are the true class labels from the dataset. Columns (Predicted) are the model predictions. Class 0 is the non-e1 cell type while Class 1 is the e1 cell type. Breaking down the matrix components, True Negatives (TN) had 568 correctly identified non-e1 samples. False Positives (FP) had 3 non-e1 samples misclassified as e1. False Negatives (FN) had 1 e1 sample misclassified as non-e1. True Positives (TP) had 297 correctly identified e1 samples.

Table 14

Final Combined Confusion Matrix				
		Predicted		
	Class	**0**	**1**	**Total**
Actual	**0**	568	3	571
	1	1	297	298
	Total	569	300	869

There are 4 misclassifications out of 869 samples providing a 0.46% error rate. The false positive rate is 0.53% (3/568 non-e1 samples misclassified). The false negative rate is 0.34% (1/298 e1 samples misclassified). This demonstrates that there is a slightly higher false positive than false negative rate. With multiple parameter combinations achieving identical performance scores of 0.99540: n_estimators = 500, 1000, 2000 with max_features = 12, this stability indicates model robustness and suggests that 500 trees is sufficient.

The OOB score was 0.99540 and the CV Accuracy was 0.99540. The difference is 0.00000 meaning perfect agreement. This agreement validates the method reliability in the sense that the OOB estimation provides an unbiased performance estimate that is equivalent to the formal cross-validation. It also means that the bootstrap sampling in Random Forest accurately represents the underlying data distribution and both methods consistently identify the same level of classification performance demonstrating model stability. With only 4 total misclassifications out of 869 training samples, the model demonstrates exceptional discriminative power between e1 and non-e1 cell types.

Based on consistent performance across both evaluation methods, the optimal model is NTREE = 500 trees, MTRY = 12 features per node, Cutoff = 0.5 (majority vote), bootstrap = True (default), and random_state = 42 (reproducibility). All accuracy measures based on the final combined confusion matrix are provided below in table 15:

Table 15

Comprehensive Performance Metrics			
Metric	**Value**	**Formula Used**	**Calculation**
Accuracy	0.99540	(TP + TN) / (TP + TN + FP + FN)	(568 + 297) / (568 + 3 + 1 + 297)
Precision	0.99000	TP / (TP + FP)	297 / (297 + 3)
Recall	0.99664	TP / (TP + FN)	297 / (297 + 1)

F1 Score	0.99331	2 x (Precision x Recall) / (Precision + Recall)	2 x (0.99000 x 0.99664) / (0.99000 + 0.99664)
OOB Score	0.99540	Out-of-bag estimation	Bootstrap sample error rate

Accuracy represents the proportion of correct predictions across all classes with it being the primary metric for overall model performance assessment. Precision measured the proportion of positive predictions that are actually correct and was critical for understanding false positive rates in the biological classification. Recall measured the proportion of actual positive cases correctly identified and was essential for assessing the model's ability to detect e1 clusters. The F1 score was a means of precision and recall and provided a balanced assessment when class distribution was uneven.

Feature Ranking

To compute feature ranking, I used Gini Importance from the best-trained OOB model: n_estimators = 500, max_features = 12; which achieved the optimal performance score of 0.99540. The Gini Importance metric represents each feature's average contribution to impurity reduction across all 500 decision trees in the Random Forest model. The features with higher importance values (ranks) contribute more to accurate classification between e1 and non-e1 cell types. The feature importance extraction was implemented as shown below in table 16:

Table 16

```
# Extract feature importances from best OOB model
importances = best_OOB_model.feature_importances_
feature_importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values('importance', ascending=False)
```

Table 17 reports the feature ranking analysis conducted using the best-trained OOB model and reports the top 10 gene expression markers. The rankings are derived from the RF Gini impurity reduction across 500 trees trained on 869 e1 cell classification samples.

Table 17

Top 10 Most Important Features (Gini Importance)		
Feature Name	**Rank**	**Importance Score**
TESPA1	1	0.02607
SLC17A7	2	0.02539

SFTA1P	3	0.02530
TBR1	4	0.02448
LINC00152	5	0.02366
LINC00507	6	0.02345
KCNIP1	7	0.02327
SLIT3	8	0.02216
LINC00710	9	0.02128
ANKRD33B	10	0.02120

The Random Forest feature ranking shows a remarkable alignment with the knowledge presented from the source research paper. The ground truth definition again states that “e1 clusters are characterized by neurons that selectively express TESPA1, LINC00507, and SLC17A7 mRNAs and lack expression of KNCP1 mRNA”. As seen in table 18, TESPA1 is Rank 1, Importance: 0.02607. This gene expression being Rank 1 perfectly corresponds with ground truth as the primary e1 marker. The model correctly identifies this as the most discriminative feature thus validating both the biological research as well as the Random Forest’s ability to interpret. Continuing on this note, SLC17A7 is Rank 2, Importance: 0.02539. This strong alignment with biological findings and ranking the second highest confirms its critical role in e1 cell type identification. LINC00507 comes in at Rank 6, Importance: 0.02345. It corresponds to the biological research as it is present in the top 10 features despite ranking slightly lower than the other two primary markers. KNCP1 is absent from the dataset entirely which is consistent with the ground truth statement that e1 cells “lack expression of KNCP1 mRNA” thus supporting the biological accuracy of the provided dataset.

Table 18

Ground Truth Comparison		
Feature Name	**Rank**	**Importance Score**
TESPA1	Rank 1	0.02607
LINC00507	Rank 6	0.02345
SLC17A7	Rank 2	0.02539
KNCP1	Not Found in Dataset	N/A

The ranking values indicate that collectively they approximately account for 24% of the total predictive power which means the characteristics are distributed rather than concentrated as a biological control. Meaning, there exists a gradual importance decline suggesting hierarchical

biological significance rather than a few dominant markers indicating that e1 cell type classification relies on multiple coordinated gene expression patterns. There is no single dominant feature as TESPA1 (Rank 1) only accounts for 2.61% of total importance further proving that accurate classification requires the integration of multiple biological signals rather than dependence on a single biomarker. Knowing this, based on the biological validation and gradual importance decline pattern, I would use the top 10-20 features. This would likely maintain a high predictive accuracy while lowering risk of overfitting in new datasets. Again, due to the gradual decline in importance values, diminishing returns beyond the top 20 features which should capture around 45% of the predictive power, eliminating a large number of less informative features.

RF Run Time Test

The optimal Random Forest Model (n_estimators = 500, max_features = 12) was used as a runtime prediction engine, simulating real-world classification scenarios. Achieving 99.54% accuracy during training and cross-validation, the model was used to classify the two verification samples that were placed in the verification database apart from the original dataset. This simulates where the model processes individual samples to provide both class predictions and confidence estimates through probability scores. The runtime prediction implementation utilized the model's prediction capabilities. *Refer to table 19.*

Table 19

```
# Runtime prediction on verification samples
X_verification = verify_db.drop('Label', axis=1)
y_true_verification = verify_db['Label']
predictions = best_OOB_model.predict(X_verification)
prediction_probs = best_OOB_model.predict_proba(X_verification)
```

Below in table 20 are the runtime prediction results from the two-sample verification database using the optimal Random Forest model (n_estimators = 500, max_features = 12, random_state = 42). The verification samples were randomly selected from the original dataset to ensure class balance. The listed probability scores represent ensemble voting across 500 trees and the confidence is calculated as maximum probability across classes.

Table 20

Verification Sample Predictions					
Sample	**True Class**	**Predicted Class**	**Prob Class 0**	**Prob Class 1**	**Correct?**
1	0	0	0.97600	0.02400	Yes
2	1	1	0.00600	0.99400	Yes

Verification accuracy was 100% as 2/2 samples were correctly classified. The average confidence was 98.5% ((97.6% + 99.4%) / 2). This classification consistency perfectly aligns with the ground truth labels. Both verification samples were correctly classified thus demonstrating the model's ability to generalize beyond the training data. For sample 1, the non-e1 cell, it was correctly identified as Class 0 with 97.6% confidence. The high probability for Class 0 (0.976) versus the low probability for Class 1 (0.0024) indicates strong discriminative capability for non-e1 cell types. For sample 2, the e1 cell, it correctly identified as Class 1 with 99.4% confidence. This extremely low probability for Class 0 (0.006) and high probability for Class 1 (0.994) demonstrates huge certainty in e1 cell identification. The prediction results align with the expected biological patterns established during feature ranking analysis. The model's high confidence in both predictions suggests that the verification samples exhibit clear gene expression signatures consistent with their cell types. This confidence is established through multiple indicators, probability, CV consistency, model stability, and biological validation alignment. As both predictions exceed 95% probability thresholds which indicate strong ensemble agreement. Meaning, Sample 1 with a 97.6% confidence represents 488+ trees out of 500 voting for Class 0. For Sample 2, 99.4% confidence represents 497+ trees out of 500 voting for Class 1. The 99.54% CV accuracy provides strong support for prediction reliability as well as it aligns with the model's established performance pattern suggesting the verification samples fall within the decision space of the trained model. The consistent performance across different parameter combinations during training with multiple configurations achieving 0.99540 accuracy also indicate robust model behavior and reflect genuine biological patterns.

Table 21

Verification Accuracy	1.00000
Prediction Confidence: Max Probabilities	[0.976 , 0.994]
Average Prediction Confidence	0.98500

Resources

- CS 486 USF Fall 2025 HW 2 Instructions ML Pipeline Experiment RF.pdf
- HW 2 Research Paper Explaining Data with Ground Ruth.pdf
- Original Training Data for HW 2 - e1 cluster data (features and ground truth e.g class label e1 (1) and not-e1 (0))
- Intro to Scikit-Learn Kalra V2 03-08-20.pdf
- RF Experiments - Useful hints for SciKit users for HW 2.docx
- Code for step by step 3 fold Cross Validation (CV) for Random Forest in SciKit
- Anaconda Assistant (twice for syntax)
- Classification and Regression Trees and Random Forest slides
- Perplexity AI on brief guide setup
- OOB Errors for Random Forest in Scikit Learn GeeksforGeeks
- Scikit-learn documentation
- Youtube: Getting Started with SciKit
    - How to use Random Forest in SciKit

- OOB Score Explained in SciKit
- Difference Between OOB and CV
- Cross Validation Explained in SciKit
- Computing Confusion Matrix, F1 scores, Precision and More in SciKit

Appendix I

I utilized Perplexity Free when initializing setting up the layout of my code. I asked for a brief outline of the flow/progression needed to complete each box on the checklist. Provided I received outlines like this:

```
# --- 1. Load and Audit Data ---
df = pd.read_csv('data.csv')
print(data shape)
print(data columns)
print("Value counts:", df['label'].value_counts())
print(df.describe())
```

This helped me get started and move in a positive direction. Feeling lost and slightly overwhelmed initially, this allowed me to conceptualize how the code will be formatted and progress, allowing me to get started. It helped highly. Another time I used Perplexity AI Free was when I was uncertain about the results of my Top 10 Feature Ranking and plugged the output into Perplexity AI asking whether or not the provided ranks looked accurate in terms of the ground truth comparison portion, typing:

```
Is this accurate?
8. Feature Ranking
------------------------------------------------------------
Top 10 Most Important Features (Gini Importance):
 1. TESPA1          0.02607
 2. SLC17A7         0.02539
 3. SFTA1P          0.02530
 4. TBR1           0.02448
 5. LINC00152       0.02366
 6. LINC00507       0.02345
 7. KCNIP1          0.02327
 8. SLIT3          0.02216
 9. LINC00710       0.02128
10. ANKRD33B        0.02120
```

Ground Truth Comparison:
Biology paper: e1 cluster 'selectively expresses TESPA1, LINC00507 and SLC17A7 mRNAs, and lacks expression of KNCP1 mRNA'
TESPA1: Rank 54 (importance: 0.02607)
LINC00507: Rank 55 (importance: 0.02345)
SLC17A7: Rank 56 (importance: 0.02539)

KNCP1: Not found in dataset

Perplexity Response:

There's a significant inconsistency in your output that indicates a coding error. The Problem: Your output shows:

- Top 10 list: TESPA1 is #1, SLC17A7 is #2, LINC00507 is #6
- Ground truth comparison: Says TESPA1 is Rank 54, SLC17A7 is Rank 56, LINC00507 is Rank 55

This is mathematically impossible. If TESPA1 is the #1 most important feature, it cannot simultaneously be ranked #54. What's Happening: There's likely a bug in the ground truth comparison section. The code is probably:

1. Correctly identifying the top 10 features
2. But incorrectly calculating the ranks in the ground truth comparison (possibly using wrong indices or DataFrame operations)

The bug is in this line: rank_in_top10 = top_10_feats[top_10_feats['feature'] == gene].index[0] + 1
The problem is that you're using .index[0], which gives you the original DataFrame index (which could be any number from 0-607), not the position within the top 10 list. Replace the line with: rank_in_top10 = top_10_feats.reset_index(drop=True)[top_10_feats['feature'] == gene].index[0] + 1

The clarification and explanation, breaking down what exactly I was looking at prevented me from staring at the issue for countless hours. Asking for advice allowed me to continue progressing in the code development rather than waiting around hoping that I would eventually see the issue. This was extremely helpful, AI response ranked high in explanation. However, the proposed fix was not the fix as a few more lines related to the proposed line were needed but it got me going on the correct track. AI just needs to be double checked before proceeding blindly. Overall, very helpful.

Appendix II

```
# CS 486 HW2 - Random Forest Classification Pipeline
# Author: Cade Mendoza
# Date: September 25, 2025
# Purpose: Complete ML pipeline for e1 cluster classification using Random Forest

import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score,
precision_score, recall_score
import warnings
warnings.filterwarnings('ignore')

# Set random seed for reproducibility across all random operations
np.random.seed(42)

print("CS 486 HW2 - Random Forest Classification Pipeline")
print("=" * 60)
print("1. Audit of Original Database")
print("-" * 60)

# Load the biological dataset - e1 cluster gene expression data
# Data Loading and initial inspection: by reading path to csv file
df = pd.read_csv("/Users/Cade/Desktop/Original training DB e1 positive(1).csv")

# check if the data is correct and loaded correctly by displaying the first few rows for data
structure verification
df.head()

# Database Audit begins:
n_samples, n_features_total = df.shape
# subtract the label column thus -1
n_features = n_features_total - 1
print(f"Number of Samples: {n_samples}")
print(f"Number of Features: {n_features}")
# Class distribution analysis - understand data balance/imbalances
print(f"Class Distribution: ")
class_count = df['Label'].value_counts().sort_index()
```

```
for label, count in class_count.items():
    print(f"     Class {label}: {count} samples ({count/n_samples*100:.1f}%)")
# Data quality check - ensure ML reliability
print(f"Missing Values: {df.isnull().sum().sum()}")
print(f"Data Types: All numerical features")
# This dataset has 1.4:1 sample to feature ratio which is below recommended 10:1
# Random Forest handles this scenario well as:
# 1. Bootstrap sampling reduces overfitting risk
# 2. MTRY (feature subsampling) at each node

print("2. Create Training and Verification Databases")
print("-" * 60)
# need to remove one positive and one negative sample for verification
# Data splitting : differs from typical 80/20 splits as designed for runtime engine testing
class_0_samples = df[df['Label'] == 0]
class_1_samples = df[df['Label'] == 1]

# Use stratified sampling with a fixed random state (42) for reproducibility
verify_0 = class_0_samples.sample(n=1, random_state=42)
verify_1 = class_1_samples.sample(n=1, random_state=42)
verify_db = pd.concat([verify_0, verify_1])

# Training database = Original - verification samples
train_db = df.drop(verify_db.index)
print(f"Training Database: {train_db.shape[0]} samples")
print(f"Verification Database: {verify_db.shape[0]} samples")

# Now we need to begin to prep the training data
# Feature matrix, target vector, store for feature ranking
X_train = train_db.drop('Label', axis=1)
y_train = train_db['Label']
feature_names = X_train.columns.tolist()

print("3. Software Tools")
print("-" * 60)
print("Jupyter notebook with scikit-learn, pandas, numpy")
print(" ")

print("4. Experimental Methods and Setup")
print("-" * 60)
```

```
# It is now time to set up the hyperparameter ranges
# NTREE (n_estimators): number of trees in the forest
# MTRY (max_features): number of features to consider at each split; critical as a RF param due to controlling tree diversity and correlation
n_estimators_range = [500, 1000, 2000]
sqrt_features = int(np.sqrt(n_features))
# 0.5 x sqrt (608) = 12; sqrt(608) = 24; 2 * sqrt(608) = 48
max_features_range = [
    int(0.5 * sqrt_features),
    sqrt_features,
    int(2 * sqrt_features)
]
print(f"n_estimators (NTREE): {n_estimators_range}")
print(f"max_features (MTRY): {max_features_range}")
print(f"cutoff: 0.5 (unable to change in scikit-learn)")
print(f"Accuracy Evaluation: OOB scoring and 3-fold CV")

print("5. RF Training with OOB Scoring")
print("-" * 60)

# Out-of-Bag (OOB) Scoring - RF's built in cross validation; provides unbiased accuracy estimate without need for separate validation database
best_OOB_score = 0
best_OOB_params = {}
best_OOB_model = None

# Now we will conduct grid search over the hyperparameter
for n_est in n_estimators_range:
    for max_feat in max_features_range:
        # We begin training RF with OOB scoring
        # Number of trees, features per node, OOB accuracy calculation enabled, reproducibility, Bootstrap sampling
        rf = RandomForestClassifier(
            n_estimators=n_est,
            max_features=max_feat,
            oob_score=True,
            random_state=42,
            bootstrap=True
        )
        # Train RF on full training dataset
```

```
        rf.fit(X_train, y_train)
        # Extract OOB score
        oob_score = rf.oob_score_
        print(f"n_est={n_est}, max_feat={max_feat}: OOB={oob_score:.5f}")
        # Track best performing model for feature ranking and runtime engine and store fully
trained model
        if oob_score > best_OOB_score:
            best_OOB_score = oob_score
            best_OOB_params = {'n_estimators': n_est, 'max_features': max_feat}
            best_OOB_model = rf

print(f"Best OOB params: n_estimators={best_OOB_params['n_estimators']},
max_features={best_OOB_params['max_features']}")
print(f"Best OOB score: {best_OOB_score:.5f}")

print("6. 3-Fold Cross Validation")
print("-" * 60)
# Stratified K-Fold CV maintains class distribution in each fold
# Reliable accuracy estimation + imbalanced data -> critical
kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

best_CV_score = 0
best_CV_params = {}
all_FOLD_CMS = []          # Store confusion matrices from all folds
all_FOLD_Predicts = []     # Store predictions for combined analysis
all_FOLD_True_labels = []  # Store true labels for combined analysis
best_fold_details = []     # Store individual fold info for best params

# It is now time to test each parameter combination (from Prof. pseudo code/lectures) using grid
search with manual CV implementation
for n_est in n_estimators_range:
    for max_feat in max_features_range:
        fold_scores = []
        fold_cms = []
        fold_predicts = []
        fold_true_labels = []
        current_fold_details = [] # track fold details for current param combination

        # it is now time to perform 3-Fold CV
        for fold_idx, (train_idx, test_idx) in enumerate(kfold.split(X_train, y_train)):
```

```
        # Split data for current fold, maintaining stratification
        X_train_fold = X_train.iloc[train_idx]
        X_test_fold = X_train.iloc[test_idx]
        y_train_fold = y_train.iloc[train_idx]
        y_test_fold = y_train.iloc[test_idx]

        # Need to now train RF on the fold training data
        rf = RandomForestClassifier(n_estimators=n_est, max_features=max_feat,
random_state=42)
        rf.fit(X_train_fold, y_train_fold)
        # Need to now test on the fold test data
        y_pred_fold = rf.predict(X_test_fold)
        fold_score = accuracy_score(y_test_fold, y_pred_fold)
        fold_scores.append(fold_score)
        # Need to now store for the final confusion matrix: predictions and labels
        fold_predicts.extend(y_pred_fold)
        fold_true_labels.extend(y_test_fold)
        # Create fold specific confusion matrix
        cm_fold = confusion_matrix(y_test_fold, y_pred_fold)
        fold_cms.append(cm_fold)

        # Store detailed fold information
        current_fold_details.append({
            'fold': fold_idx + 1,
            'accuracy': fold_score,
            'confusion_matrix': cm_fold,
        })
    # Average accuracy across all folds for this parameter combination
    avg_CV_score = np.mean(fold_scores)
    print(f"n_est={n_est}, max_feat={max_feat}: CV={avg_CV_score:.5f}")
    # Track best performing parameter combination
    if avg_CV_score > best_CV_score:
        best_CV_score = avg_CV_score
        best_CV_params = {'n_estimators': n_est, 'max_features': max_feat}
        all_FOLD_CMS = fold_cms
        all_FOLD_Predicts = fold_predicts
        all_FOLD_True_labels = fold_true_labels
        best_fold_details = current_fold_details # save fold details for best params
```

```
print(f"Best CV params: n_estimators={best_CV_params['n_estimators']},
max_features={best_CV_params['max_features']}")
print(f"Best CV score: {best_CV_score:.5f}")

# print individual fold confusion matrices
print(f"\nIndividual Fold Analysis for Optimal Parameters")
print(f"Parameters: n_estimators = {best_CV_params['n_estimators']},
max_features={best_CV_params['max_features']}")
print("-" * 60)
# print detailed confusion matrix for each fold
for fold_detail in best_fold_details:
    cm = fold_detail['confusion_matrix']
    accuracy = fold_detail['accuracy']
    print(f"\nFold {fold_detail['fold']} Confusion Matrix:")
    print(f"          Predicted")
    print(f"Actual     0     1     Total")
    print(f"   0    {cm[0,0]:4d}    {cm[0,1]:3d}     {cm[0,:].sum()}")
    print(f"   1    {cm[1,0]:4d}    {cm[1,1]:3d}     {cm[1,:].sum()}")
    print(f"Total    {cm[:,0].sum():4d}    {cm[:,1].sum():3d}     {cm.sum()}")
    print(f"Accuracy: {accuracy:.5f} ({cm[0,0] + cm[1,1]}/{cm.sum()} correct)")

print("7. Results of RF Training and Accuracy Estimates")
print("-" * 60)

# Now it is time to combine the confusion matrix from all CV folds for final assessment
# This provides accuracy metrics based on all the training data
combined_CM = np.sum(all_FOLD_CMS, axis=0)

# Calculate the comprehensive accuracy metrics
accuracy = accuracy_score(all_FOLD_True_labels, all_FOLD_Predicts)
precision = precision_score(all_FOLD_True_labels, all_FOLD_Predicts)
recall = recall_score(all_FOLD_True_labels, all_FOLD_Predicts)
f1 = f1_score(all_FOLD_True_labels, all_FOLD_Predicts)
# Display the optimal RF parameters
print(f"Best RF Params:")
print(f"    NTREE (n_estimators): {best_CV_params['n_estimators']}")
print(f"    MTRY (max_features): {best_CV_params['max_features']}")
print(f"    CUTOFF: 0.5")
# Display the final combined confusion matrix in professional format
print(f"\nFinal Combined Confusion Matrix: ")
```

```
print(f"                  Predicted")
print(f"        Class      0        1        Total")
print(f"Actual     0     {combined_CM[0,0]:4d}     {combined_CM[0,1]:4d}
{combined_CM[0,:].sum():4d}")
print(f"          1     {combined_CM[1,0]:4d}     {combined_CM[1,1]:4d}
{combined_CM[1,:].sum():4d}")
print(f"        Total   {combined_CM[:,0].sum():4d}     {combined_CM[:,1].sum():4d}
{combined_CM.sum():7d}")
# Report the comprehensive accuracy metrics all with 5 decimal precision
print(f"\nAccuracy Measures:")
print(f"  Accuracy:  {accuracy:.5f}")         # (TP + TN) / Total
print(f"  Precision: {precision:.5f}")        # TP / (TP + FP)
print(f"  Recall:    {recall:.5f}")           # TP / (TP + FN)
print(f"  F1 Score:  {f1:.5f}")               # mean of precision/recall
print(f"  OOB Score: {best_OOB_score:.5f}")    # compare with CV results

print("8. Feature Ranking")
print("-" * 60)

# Now it is time to use the best OOB model to conduct a feature ranking (trained on the full data)
for most reliable ranking
# Feature importance calculation using Gini Importance - critical for RF explainability as it
shows which genes drive classification
importances = best_OOB_model.feature_importances_
# Create ranked feature importance dataframe
feature_importance_df = pd.DataFrame({
   'feature': feature_names,
   'importance': importances
}).sort_values('importance', ascending=False)
# Extract top 10 features for analysis
top_10_feats = feature_importance_df.head(10)

print("Top 10 Most Important Features (Gini Importance): ")
print("Rank  Feature Name   Importance")
print("-" * 60)
for idx, (_, row) in enumerate(top_10_feats.iterrows(), 1):
   print(f"{idx:2d}     {row['feature']:15s}  {row['importance']:.5f}")

# Now it is time to compare with the biological ground truth
ground_truth_GENES = ['TESPA1', 'LINC00507', 'SLC17A7', 'KNCP1']
```

```
print(f"\nGround Truth Comparison:")
print(f"Biology paper: e1 cluster 'selectively expresses TESPA1, LINC00507 and SLC17A7 mRNAs, and lacks expression of KNCP1 mRNA'")

# Create a reset index version of the dataframe for easier ranking and accurate ranking calculation
feature_importance_reset = feature_importance_df.reset_index(drop=True)

# Compare RF rankings with established biological knowledge
# From the next line to the line print (f"  {gene}: Not found in dataset") the code was pseudocoded from Perplexity
for gene in ground_truth_GENES:
    if gene in top_10_feats['feature'].values:
        # gene found in top 10, then calculate its rank
        rank_in_top10 = top_10_feats.reset_index(drop=True)[top_10_feats.reset_index(drop=True)['feature'] == gene].index[0] + 1
        importance = top_10_feats[top_10_feats['feature'] == gene]['importance'].iloc[0]
        print(f"  {gene}: Rank {rank_in_top10} (importance: {importance:.5f})")
    else:
        # Gene not in top 10 - find its actual rank in full feature set
        gene_row = feature_importance_reset[feature_importance_reset['feature'] == gene]
        if not gene_row.empty:
            actual_rank = gene_row.index[0] + 1  # Get the index directly from the reset dataframe
            importance = gene_row['importance'].iloc[0]
            print(f"   {gene}: Rank {actual_rank} (importance: {importance:.5f}) - not in top 10")
        else:
            print(f"  {gene}: Not found in dataset")

print("9. RF Runtime Test")
print("-" * 60)

# Now it is time to predict the verification samples using the best trained model (OOB) trained on full training data
# Feature matrix for verification and ground truth labels
X_verification = verify_db.drop('Label', axis=1)
y_true_verification = verify_db['Label']
# Generate predictions using trained RF runtime engine: class decisions and class probabilities
predictions = best_OOB_model.predict(X_verification)
prediction_probs = best_OOB_model.predict_proba(X_verification)
```

```
print("Verification Sample Predictions: ")
print("Sample  True  Predicted  Prob_0  Prob_1  Correct?")
for i in range(len(X_verification)):
    true_class = y_true_verification.iloc[i]
    pred_class = predictions[i]
    prob_0 = prediction_probs[i][0]      # Probability of class 0
    prob_1 = prediction_probs[i][1]      # Probability of class 1
    correct = "Yes" if true_class == pred_class else "No"
    print(f"{i+1:1d}   {true_class:6d}      {pred_class:1d}          {prob_0:.5f}  {prob_1:.5f}
{correct:8s}")
verify_accuracy = accuracy_score(y_true_verification, predictions)
print(f"\nVerification Accuracy: {verify_accuracy:.5f}")

# Now we must do a confidence assessment: based on prediction probabilities
max_probs = np.max(prediction_probs, axis=1)
print(f"Prediction Confidence: Max Probabilities = {max_probs}")
average_confidence = np.mean(max_probs)
print(f"Average Prediction Confidence: {average_confidence:.5f}")

print("\n" + "=" * 60)
print("Experiment Complete")
print("=" * 60)
print(f"Summary:")
print(f"   Best OOB Score: {best_OOB_score:.5f}")          # RF built in CV result
print(f"   Best CV Score:  {best_CV_score:.5f}")           # manual 3-fold CV result
print(f"   Final Accuracy: {accuracy:.5f}")                # combined fold accuracy
print(f"   F1 Score:       {f1:.5f}")                      # balanced accuracy measure
print(f"   Verification:   {verify_accuracy:.5f}")         # runtime engine test

# OOB and CV scores match exactly 0.99540
# High accuracy 99.54% confirms dataset's strong biological signal
# Feature Ranking shows relevance to source research paper
# Verification Accuracy with high confidence 97.6% and 99.4%
```