In [2]:
```
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')
```
In [3]:
```
np.random.seed(42)
```
In [4]:
```
print("CS 486 HW2 - Random Forest Classification Pipeline")
print("=" * 60)
print("1. Audit of Original Database")
print("-" * 60)
```
```
CS 486 HW2 - Random Forest Classification Pipeline
============================================================
1. Audit of Original Database
------------------------------------------------------------
```
In [5]:
```
# Load data by reading path to csv file
df = pd.read_csv("/Users/Cade/Desktop/Original training DB e1 positive(1).csv")
```
In [6]:
```
# check if the data is correct and loaded correctly
df.head()
```
Out[6]:

	GABRG2	CELF4	SRRM4	SLC1A3	ATP1A3	RBFOX3	GABRA4	NH!
0	35.038262	161.176004	68.074337	58.063405	20.021864	269.294069	188.205520	0.0000
1	95.324867	75.256474	87.297510	0.000000	18.061554	342.166102	683.328784	0.0000
2	220.143867	187.976727	42.219372	106.553653	0.000000	187.976727	299.556496	0.0000
3	166.010840	26.159284	61.373704	0.000000	30.183789	254.549955	446.720079	0.0000
4	188.426220	71.160966	119.269788	57.129226	16.036274	265.600789	287.650666	24.054

5 rows × 609 columns

In [7]:
```
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}")
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}%)")

print(f"Missing Values: {df.isnull().sum().sum()}")
print(f"Data Types: All numerical features")
```

```
# check to make sure I need some of these print statements and whether I need to say I used AI for this
```

```
Number of Samples: 871
Number of Features: 608
Class Distribution:
    Class 0: 572 samples (65.7%)
    Class 1: 299 samples (34.3%)
Missing Values: 0
Data Types: All numerical features
```

In [8]:

```
print("2. Create Training and Verification Databases")
print("-" * 60)
# need to remove one positive and one negative sample for verification
class_0_samples = df[df['Label'] == 0]
class_1_samples = df[df['Label'] == 1]

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])

train_db = df.drop(verify_db.index)
print(f"Training Database: {train_db.shape[0]} samples")
print(f"Verification Databse: {verify_db.shape[0]} samples")

# Now we need to begin to prep the training data
X_train = train_db.drop('Label', axis=1)
y_train = train_db['Label']
feature_names = X_train.columns.tolist()
```

```
2. Create Training and Verification Databases
------------------------------------------------------------
Training Database: 869 samples
Verification Databse: 2 samples
```

In [9]:

```
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 param ranges
# NTREE
n_estimators_range = [500, 1000, 2000]
sqrt_features = int(np.sqrt(n_features))
# 0.5 x sqrt (608) = 12; sqrt(608) = 25; 2 * sqrt(608) = 49
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")
```

```
3. Software Tools
------------------------------------------------------------
Jupyter notebook with scikit-learn, pandas, numpy
```

4. Experimental Methods and Setup
----
n_estimators (NTREE): [500, 1000, 2000]
max_features (MTRY): [12, 24, 48]
cutoff: 0.5 (unable to change in scikit-learn)
Accuracy Evaluation: OOB scoring and 3-fold CV

In [10]:

```
print("5. RF Training with OOB Scoring")
print("-" * 60)
best_OOB_score = 0
best_OOB_params = {}
best_OOB_model = None

for n_est in n_estimators_range:
    for max_feat in max_features_range:
        # We begin training RF with OOB scoring
        rf = RandomForestClassifier(
            n_estimators=n_est,
            max_features=max_feat,
            oob_score=True,
            random_state=42,
            bootstrap=True
        )
        rf.fit(X_train, y_train)

        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

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}")
```

```
5. RF Training with OOB Scoring
------------------------------------------------------------
n_est=500, max_feat=12: OOB=0.99540
n_est=500, max_feat=24: OOB=0.99540
n_est=500, max_feat=48: OOB=0.99310
n_est=1000, max_feat=12: OOB=0.99540
n_est=1000, max_feat=24: OOB=0.99540
n_est=1000, max_feat=48: OOB=0.99310
n_est=2000, max_feat=12: OOB=0.99540
n_est=2000, max_feat=24: OOB=0.99540
n_est=2000, max_feat=48: OOB=0.99310
Best OOB params: n_estimators=500, max_features=12
Best OOB score: 0.99540
```

In [22]:

```
print("6. 3-Fold Cross Validation")
print("-" * 60)
kfold = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
best_CV_score = 0
best_CV_params = {}
all_FOLD_CMS = []
```

```
all_FOLD_Predicts = []
all_FOLD_True_labels = []
best_fold_details = [] # store individual fold info for best params
# It is now time to test each paramater combination (from Prof. pseudo code/lectures)
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 params

        # it is now time to perform 3-Fold CV
        for fold_idx, (train_idx, test_idx) in enumerate(kfold.split(X_train, y_train)):
            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
            fold_predicts.extend(y_pred_fold)
            fold_true_labels.extend(y_test_fold)
            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,
            })

        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
            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)

for fold_detail in best_fold_details:
    cm = fold_detail['confusion_matrix']
    cm_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)")
```

```
6. 3-Fold Cross Validation
------------------------------------------------------------
n_est=500, max_feat=12: CV=0.99540
n_est=500, max_feat=24: CV=0.99424
n_est=500, max_feat=48: CV=0.99195
n_est=1000, max_feat=12: CV=0.99540
n_est=1000, max_feat=24: CV=0.99540
n_est=1000, max_feat=48: CV=0.99195
n_est=2000, max_feat=12: CV=0.99540
n_est=2000, max_feat=24: CV=0.99425
n_est=2000, max_feat=48: CV=0.99195
Best CV params: n_estimators=500, max_features=12
Best CV score: 0.99540

Individual Fold Analysis for Optimal Parameters
Parameters: n_estimators = 500, max_features=12
------------------------------------------------------------

Fold 1 Confusion Matrix:
          Predicted
Actual     0     1     Total
   0     189     1     190
   1       0   100     100
Total    189   101     290
Accuracy: 0.99540 (289/290 correct)

Fold 2 Confusion Matrix:
          Predicted
Actual     0     1     Total
   0     189     2     191
   1       0    99     99
Total    189   101     290
Accuracy: 0.99540 (288/290 correct)

Fold 3 Confusion Matrix:
          Predicted
Actual     0     1     Total
   0     190     0     190
   1       1    98     99
Total    191    98     289
Accuracy: 0.99540 (288/289 correct)
```

In [32]:

```
print("7. Results of RF Training and Accuracy Estimates")
```

```
print("-" * 60)

# Now it is time to combine the confusion matrix from CV
combined_CM = np.sum(all_FOLD_CMS, axis=0)
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)

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")

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}")

print(f"\nAccuracy Measures:")
print(f"  Accuracy:  {accuracy:.5f}")
print(f"  Precision: {precision:.5f}")
print(f"  Recall:    {recall:.5f}")
print(f"  F1 Score:  {f1:.5f}")
print(f"  OOB Score: {best_OOB_score:.5f}")
```

```
7. Results of RF Training and Accuracy Estimates
------------------------------------------------------------
Best RF Params:
    NTREE (n_estimators): 500
    MTRY (max_features): 12
    CUTOFF: 0.5

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

Accuracy Measures:
  Accuracy:  0.99540
  Precision: 0.99000
  Recall:    0.99664
  F1 Score:  0.99331
  OOB Score: 0.99540
```

In [33]:

```
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)
importances = best_OOB_model.feature_importances_
```

```
feature_importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values('importance', ascending=False)

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 comprare 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
feature_importance_reset = feature_importance_df.reset_index(drop=True)

for gene in ground_truth_GENES:
    if gene in top_10_feats['feature'].values:
        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:
        # Find the position in the full list
        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")
```

```
8. Feature Ranking
------------------------------------------------------------
Top 10 Most Important Features (Gini Importance):
Rank  Feature Name   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 express
```

```
ion of KNCP1 mRNA'
  TESPA1: Rank 1 (importance: 0.02607)
  LINC00507: Rank 6 (importance: 0.02345)
  SLC17A7: Rank 2 (importance: 0.02539)
  KNCP1: Not found in dataset
```

In [34]:

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

# Now it is time to predict the verification samples using the best trained model
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)

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]
    prob_1 = prediction_probs[i][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
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}")
print(f"   Best CV Score:  {best_CV_score:.5f}")
print(f"   Final Accuracy: {accuracy:.5f}")
print(f"   F1 Score:       {f1:.5f}")
print(f"   Verification:   {verify_accuracy:.5f}")
```

```
9. RF Runtime Test
------------------------------------------------------------
Verification Sample Predictions: 
Sample   True   Predicted   Prob_0   Prob_1   Correct?
1        0      0           0.97600  0.02400  Yes
2        1      1           0.00600  0.99400  Yes

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

============================================================
Experiment Complete
```

```
============================================================
Summary:
  Best OOB Score: 0.99540
  Best CV Score:  0.99540
  Final Accuracy: 0.99540
  F1 Score:       0.99331
  Verification:   1.00000
```