A simple ML experiment with Random Forest ML (RF) using SciKit SW Toolkits and ChatGPT "adviser"

859 2024Fall 10/20/2024 Zoe Long

1. Audit of Training database

1.1 Describe source of Data

The emergence of single-cell transcriptional profiling through RNA sequencing is generating much more information than before, which also has resulted in challenges for representation in the Cell Ontology.

The data in file "e1 positive.csv" comes from single nuclei extracted from the MTG, cortical layer 1 of a post-mortem human brain specimen and is a transcriptomic dataset obtained through RNA sequencing. The original data utilized iterative clustering method and expression of the previously characterized marker genes to identify 16 distinct cell types.

In the "e1 positive.csv" file, only one cell type—e1—is labeled, identified by the last column. For cluster e1, the cell type is defined as "A human MTG cortical layer 2 excitatory neuron that selectively expresses TESPA1, LINC00507 and SLC17A7 mRNAs, and lacks expression of KCNIP1 mRNA." Label =1 means this instance belongs to the e1 cluster data, and a label of 0 indicates otherwise.

1.2 Audit DB

1.2.1 How are feature data obtained and their meaning

The feature data is obtained through single-cell transcriptional profiling, a groundbreaking method that allows researchers to understand the unique gene expression patterns in each cell. The DB contains transcriptomic data with 608 features and 871 instances. Each feature represents a specific gene, and the corresponding values indicate its expression levels. Variations in gene expression can lead to different cell functions and are also key factors in reference nomenclature.

1.2.2 How are class labels obtained

In [1], they identified 16 distinct cell types by clustering the resultant transcriptional profiles, with excitatory 1(e1), our label, being one of the 16 cell types. With label 1 or 0, we can know whether an instance belongs to the cluster e1.

1.2.3 Number of samples in each class and is the data unbalanced

We have 608 features and 871 instances in the dataset and about 34.3% of instances belong to the e1 cluster.

1.2.4 Type of features and missing values.

All features are numerical, with no missing value.

1.2.5 Are there enough samples compared to number of features used

Compared to the 608 features, the sample size (871) is relatively small. In general, the number of instances in a dataset should ideally be ten times or more than the number of features. However, since not all genes will have a significant impact on the classification, the number of effective features is likely to be much smaller.

1.2.6 List and description of features, formats are well documented

Each feature stands for one gene, and we have their names well documented in the csv.

2. SW tools

I used Python for the experiment. Table 1 shows all the packages I imported and the corresponding usage.

Table 1 SW Tools

Package	Function	Example
pandas	Load and preprocess dataset, pandas presents data in the form of DataFrame, which facilitates the processing of tabular data	`raw_data = pd.read_csv('datasets/e1 positive.csv')`

Scikit-learn	Scikit-learn provides a rich set of ML tools, we can directly use it for tasks like data splitting, model training and result analysis	`from sklearn.ensemble import RandomForestClassifier`
		`from sklearn.model_selection import cross_val_score`
		`from sklearn.metrics import confusion_matrix`

seaborn	Seaborn is built on top of Matplotlib and provides a concise and elegant approach to data visualization	`plt.figure(figsize=(10, 7))` `sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='ocean_r')`

matplotlib	A powerful data visualization package that allows for various customizations.	`# Add labels and title`
		`plt.xlabel('Predicted Labels')`
		`plt.ylabel('True Labels')`
		`plt.title('Confusion Matrix')`

3. Experimental Methods and Setup

3.1 Parameters and ranges of hyperparameters

To achieve the best results, I performed grid search to identify the most suitable hyperparameters. The hyperparameters tested are listed in Table 2.

Table 2 Hyperparameters

Tested Hyperparameters	
Ntree	500, 1000, 1500, 2000, 2500
Mtry	50, 100, 200, 300, 400, 500, ‘log2’, ‘sqrt’
Cutoff Range	0.5

3.2 Estimate Accuracy

To estimate accuracy, I used out-of-bag (OOB) and cross-validation for the training set, and a confusion matrix for the test set.

Since I used Random Forest (RF) in scikit-learn, I set oob_score=True to enable OOB evaluation and used sklearn.model_selection.cross_val_score to perform cross-validation (CV). The corresponding code is shown below.

```
model = RandomForestClassifier(n_estimators=ntree,
max_features=mtry, oob_score=True, random_state=98)

cv_scores = cross_val_score(model, x, y, cv=3)
```

3.3 Code Flow

Table 3 Code Flow

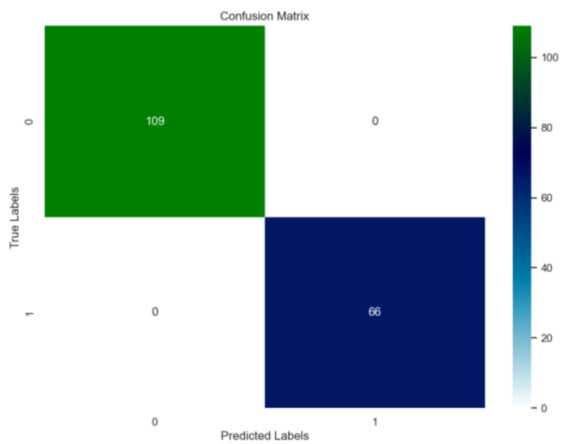
Step	Description
1	Load dataset and split data
2	Define hyperparameter ranges and initialize results dictionary
3	Train models and record results
4	Save results to JSON file
5	Read results from JSON file and find the best hyperparameter combination
6	Visualize the best results
7	Retrain model with best hyperparameters and visualize decision trees

4. Result of RF Training and Accuracy Estimates

4.1 Results for Best Trained RF

During the grid search, I stored all the results in Result.json. After analyzing the JSON file, I selected the best model based on the two metrics introduced in the following section, and both metrics identified the same model. The results of the best-trained RF are presented in Table 5.

Tabel 4 Results for Best Trained RF

Hyperparameters	Ntree:500, Mtry:50
OOB score	0.9928
Mean cross-validation	0.9931
Accuracy (on test data)	1.0
F1 Score (on test data)	1.0
Confusion_matrix	[109, 0], [0, 66]
Confusion_matrix_plot	
Top_10_Features	{'53': 'TESPA1', '55': 'SLC17A7', '54': 'LINC00507', '57': 'LINC00508', '61': 'NPTX1', '64': 'KCNIP1', '60': 'TBR1', '62': 'LINC00710', '58': 'ANKRD33B', '59': 'LINC00152'}

4.2 Measure used to optimize RF

I used grid search to optimize RF, corresponding demo is showed below.

```
ntrees = [500, 1000, 1500, 2000, 2500]
mtrys = [50, 100, 200, 300, 400, 500, 'log2','sqrt']
Result = {}

for ntree in ntrees:
    for mtry in mtrys:
        TRAINING MODEL...
```

```
STORE RESULT
```

After obtaining the results, I analyzed each RF model and selected the best one. I have two selection criteria. For the first one, I just select the model with the highest OOB, CV and F1 score. For the second one, I prioritize the feature ranking, I check if the feature ranking aligned with the definition of e1 and among the RF models that met this criterion, I chose the one with the highest accuracy. Based on these two criteria, the same model was found, with Ntree set to 500 and Mtry set to 50. The corresponding demo is shown below.

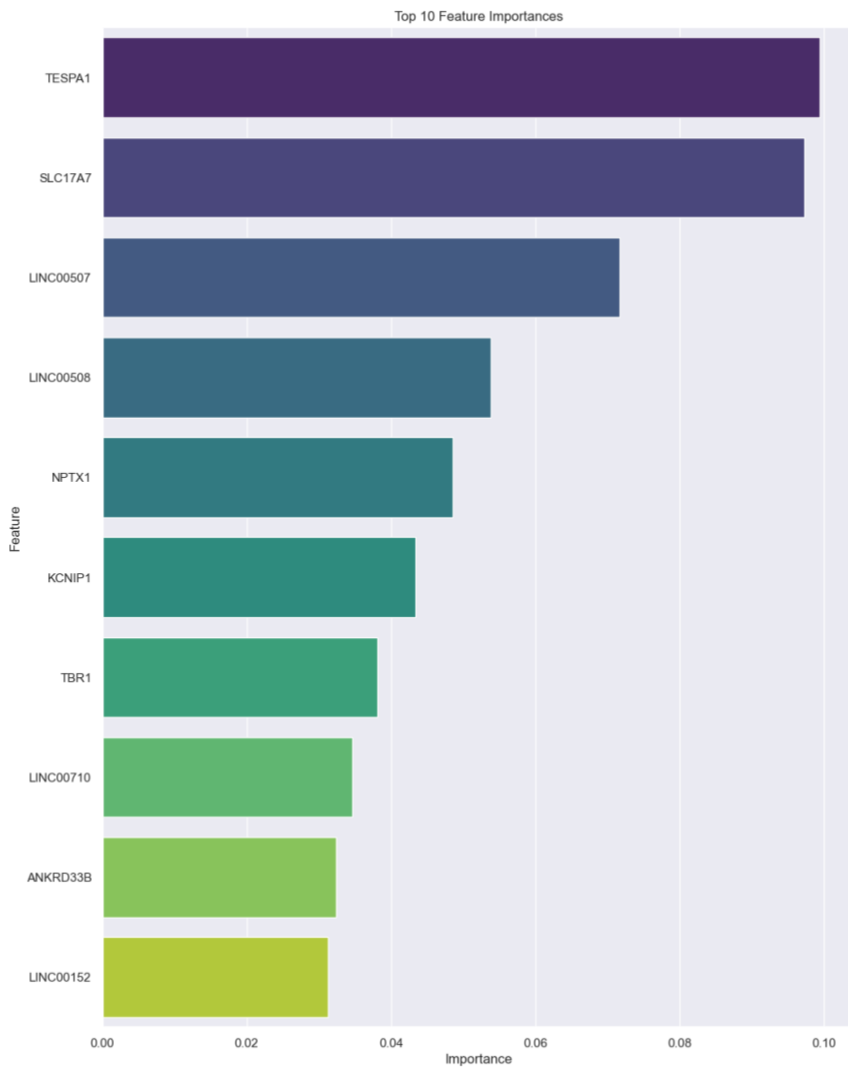
```
# Only use oob, cv and f1
for key, metrics in results.items():
    required_features = {'TESPA1', 'LINC00507', 'SLC17A7', 'KCNIP1'}
# Check if the Top 10 Features include all required features
    if required_features.issubset(top_10_features):
# Compare the current key's metrics with the best metrics
        if metrics > best_metrics:
            best_key = key
            best_metrics = metrics
```

```
# priority Feature Ranking
for key, metrics in results.items():
required_features = {'TESPA1', 'LINC00507', 'SLC17A7', 'KCNIP1'}
# Check if the Top 10 Features include all required features
if required_features.issubset(top_10_features):
# Compare the current key's metrics with the best metrics
        if metrics > best_metrics:
            best_key = key
            best_metrics = metrics
```

5. Feature Ranking

5.1 Feature Ranking for Top 10 Ranged Features

The importance of a feature is measured by the total reduction in impurity it contributes, averaged across all trees in the forest. Scikit-learn uses the Mean Decrease Impurity (MDI) method for this purpose, which calculates the decrease in impurity each time a feature is used to split a node. This information can be accessed directly through the feature_importances_ attribute of the RandomForestClassifier. Picture 1 is the visualization for the Feature Importance.



Picture1 Top 10 Feature Ranking

5.2 How does it help you in explaining how RF worked

Feature ranking helps me identify which features are more important and play a dominant role. This is valuable for explainability, as it allows me to understand the underlying logic and decision-making process of the Random Forest model.

5.3 What did I learn doing this step

The task in this paper is relatively simple, so all the tested hyperparameters yielded high accuracy. However, when examining the feature rankings, I noticed that despite the high accuracy, not all models provided reasonable rankings. This indicates that some models are not explainable, as they rely on incorrect features.

From this, I learned that accuracy alone is not sufficient to assess the quality of a model. There are many other aspects that need to be explored and explained to ensure the model's reliability and interpretability.

5.4 How it compares with ground truth established in biology paper

It is quite accord with the definition of e1, with TESPA1, LINC00507 and SLC17A7 being the top 3 and KCNIP1 the sixth.

5.4 Do ranking values show there is obvious cluster of high raked features that dominates ranking?

Yes, the importance of TESPA1, LINC00507, and SLC17A7 is higher than that of the other features, making them the dominant factors in the ranking.

5.5 How many top ranked features would you suggest be used in production (tradeoff between smaller number of features used vs. accuracy reduction)?

I'd recommend selecting the top 6 features, ending with KCNIP1. Since TESPA1, LINC00507, SLC17A7, and KCNIP1 are already known to be the key genes defining e1, adding more features might increase accuracy but could also lead to overfitting, potentially steering the classification in the wrong direction.

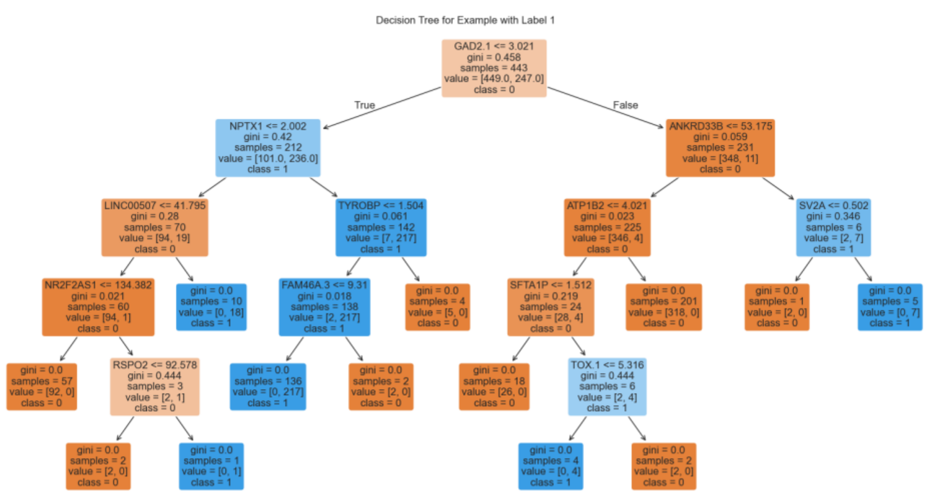
6. RF Run Time test

For this part, I selected one positive and one negative sample from the test data and fed them into the best RF model obtained earlier. Instead of directly outputting the results, I used `predict_proba` to display the proportion of trees voting for class 0 or 1. The corresponding code and output is shown below.

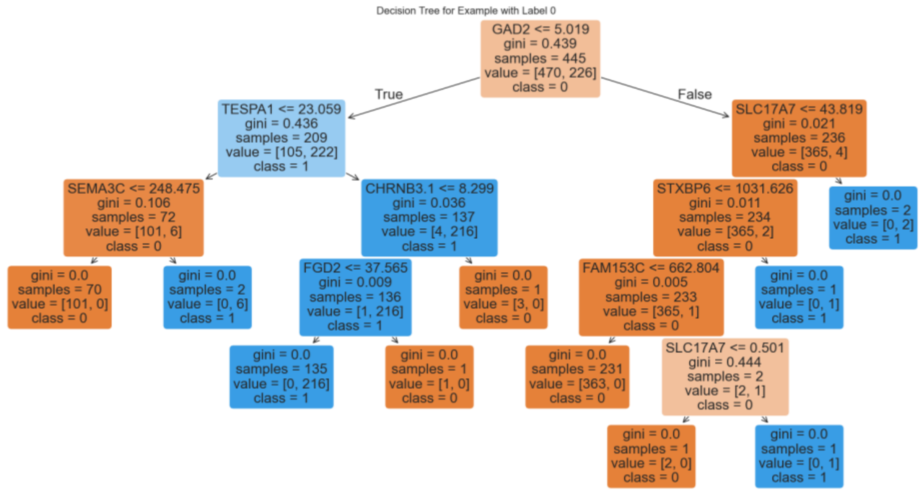
```
# Find one example each for label 1 and label 0
example_1 = x_test[y_test == 1].iloc[0]
example_0 = x_test[y_test == 0].iloc[0]
p_pre = model.predict_proba([example_1, example_0])
print(p_pre)

[[0.004 0.996]

 [0.934 0.066]]
```

According to the output above, both examples were correctly classified with high voting scores of 0.996 and 0.934. In scikit's Random Forest, the default confidence score is 0.5. However, for the two test cases above, over 90% of the trees agreed on the result, exceeding the 50% confidence threshold, which led to the correct answer.

In this section, I also visualized the decision-making processes of two trees out of the 500, even though the features used in these two trees did not rank highly in the final feature ranking. This visualization helps me better understand the decision-making processes of the trees.



Picture2 Visualization of Tree 1



Picture3 Visualization of Tree 2

7. Resources

Paper: [1] Cell type discovery using single-cell transcriptomics: implications for ontological representation

Tools: Copilot, ChatGPT

Appendix 1

Copilot:

I primarily use Copilot for code that I am unsure how to write, and I also ask it to analyze and give comment on my code. It has been very helpful, so I rely on it frequently and can't keep track of every question I ask. Below are some of my recent questions, summarized by Copilot.

I believe the key to using Copilot effectively is not to rely entirely on it. You need to understand the overall framework first, and then you can ask it to help with specific components. As long as you understand the purpose of the code and how the parts connect, Copilot can significantly boost your efficiency.

Prompt: please help me summarize all the questions I have asked for RandomForest.ipynb
Summary of Questions Asked for `RandomForest.ipynb` 1. **Finding Examples with Specific Labels and Visualizing Classification Process:** - **Question:** How to find one example each from `x_test` where the corresponding `y_test` label is 1 and 0, and visualize their classification process in the model? - **Response:** Provided pseudocode and Python code to identify examples and visualize the decision process using `plot_tree` from `sklearn.tree`. 2. **Setting Output of RandomForest to Show Probabilities:** - **Question:** How to set the output of the RandomForest model to show probabilities for each class (fraction of trees voting for 0 or 1)? - **Response:** Provided pseudocode and Python code to train the `RandomForestClassifier` and use the `predict_proba` method to get class probabilities. 3. **Organizing Code Flow and Logic into a Table:** - **Question:** How to organize the entire code flow and logic into a table and summarize the purpose of the code? - **Response:** Provided a table summarizing each step of the code along with code snippets and a brief description of the code's purpose.

ChatGPT:

Since I am not a good writer, I usually ask ChatGPT to rewrite sentences I’m not satisfied with, and it always provides great alternatives.

Appendix 2

In [16]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
sns.set()

# Load the dataset and split it
raw_data = pd.read_csv('datasets/e1 positive.csv')
feature_num = raw_data.shape[1]-1
instance_num = raw_data.shape[0]
print(feature_num)
print(instance_num)

y = raw_data['Label']
x = raw_data.drop('Label', axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rar
raw_data.describe(include='all')
```

```
608
871
```

Out[16]:

	GABRG2	CELF4	SRRM4	SLC1A3	ATP1A3	RBFOX
count	871.000000	871.000000	871.000000	871.000000	871.000000	871.00000
mean	317.195147	160.794717	188.372969	263.524808	79.262833	137.2752
std	378.241239	189.064895	208.232294	999.259112	127.012517	264.2139
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	38.894340	32.084673	35.094833	0.000000	4.019327	14.2881
50%	218.427681	108.811797	130.794368	0.000000	29.073964	73.9626
75%	440.087971	218.480061	265.911406	6.031598	100.142130	169.8164
max	3435.333490	2011.629811	1704.202638	10950.762140	1293.563390	6061.3079

8 rows × 609 columns

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix,accuracy_score
import matplotlib.pyplot as plt
import json

ntrees = [500, 1000, 1500, 2000, 2500]
mtrys = [50, 100, 200, 300, 400, 500, 'log2','sqrt']
Result = {}

for ntree in ntrees:
    for mtry in mtrys:
        model = RandomForestClassifier(n_estimators=ntree, max_features=mtry
        model.fit(x_train, y_train)
        print(f'======================Ntree:{ntree}, Mtry:{mtry}===========
        print('OOB Score: ', model.oob_score_)
```

```
        cv_scores = cross_val_score(model, x, y, cv=3)
        print("Cross-validation scores: ", cv_scores)
        print("Mean cross-validation score: ", cv_scores.mean())

        y_pred = model.predict(x_test)
        conf_matrix = confusion_matrix(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)
        print(f'Accuracy: {accuracy}')

        # Get feature importances
        feature_importances = model.feature_importances_
        # Create a DataFrame to store feature names and their importances
        feature_importance_df = pd.DataFrame({
            'Feature': x.columns,
            'Importance': feature_importances
        })
        # Sort the DataFrame by feature importances in descending order
        feature_importance_df = feature_importance_df.sort_values(by='Import
        # Select the top 10 features
        top_10_features = feature_importance_df.head(10)

        # Store the results
        Result[f"Ntree:{ntree}, Mtry:{mtry}"] = {'OOB Score': model.oob_scor
```

In [21]:

```
# Save results to JSON file
with open('result.json', 'w') as f:
    for key in Result:
        Result[key]['Confusion_matrix'] = Result[key]['Confusion_matrix'].tc
    json.dump(Result, f)
```

In [2]:

```
# Find the best key and its metrics
import json
def calculate_f1_score(confusion_matrix):
    tp = confusion_matrix[0][0]  # True Positives
    fn = confusion_matrix[0][1]  # False Negatives
    fp = confusion_matrix[1][0]  # False Positives
    tn = confusion_matrix[1][1]  # True Negatives

    # Calculate precision and recall
    precision = tp / (tp + fp) if (tp + fp) != 0 else 0
    recall = tp / (tp + fn) if (tp + fn) != 0 else 0

    # Calculate F1 score
    f1_score = 2 * (precision * recall) / (precision + recall) if (precisior

    return f1_score

with open('result.json', 'r') as f:
    results = json.load(f)

# give priority to feature ranking
# Initialize variables to store the best key and its metrics
best_key = None
best_metrics = {'OOB Score': 0, 'Mean cross-validation score': 0, 'Accuracy'
```

```
# Iterate through each key in the JSON data
for key, metrics in results.items():
    top_10_features = set(metrics['Top 10 Features']['Feature'].values())
    required_features = {'TESPA1', 'LINC00507', 'SLC17A7', 'KCNIP1'}

    # Check if the Top 10 Features include all required features
    if required_features.issubset(top_10_features):
        # Compare the current key's metrics with the best metrics
        if (metrics['OOB Score'] > best_metrics['OOB Score'] and
            metrics['Mean cross-validation score'] > best_metrics['Mean cros
            metrics['Accuracy'] > best_metrics['Accuracy']):
            best_key = key
            best_metrics = metrics

# Print the best key and its metrics
print(f"Best Key: {best_key}")
print(f"Metrics: {best_metrics}")
```

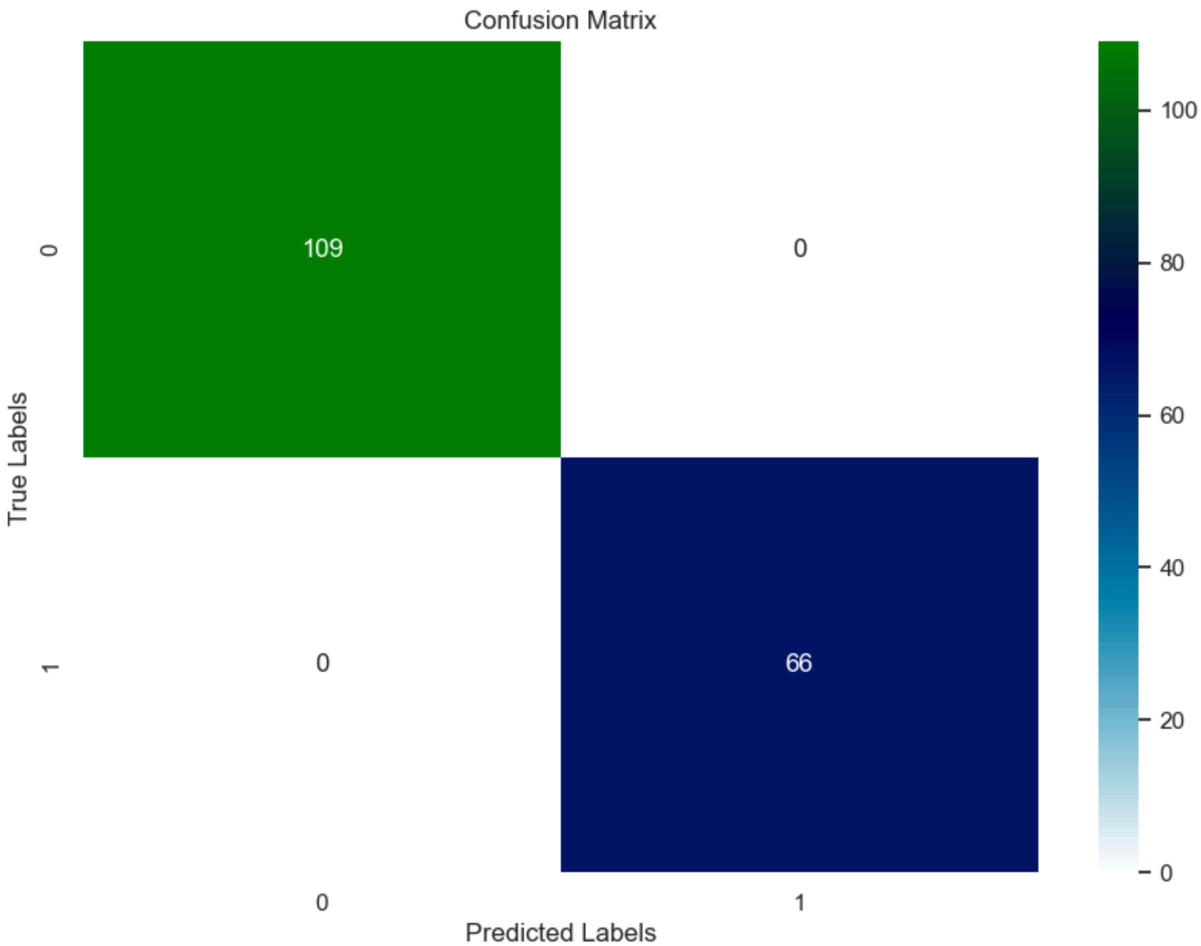
```
Best Key: Ntree:500, Mtry:50
Metrics: {'OOB Score': 0.992816091954023, 'Mean cross-validation score': 0.9
931152980210926, 'Accuracy': 1.0, 'Confusion_matrix': [[109, 0], [0, 66]],
'Top 10 Features': {'Feature': {'53': 'TESPA1', '55': 'SLC17A7', '54': 'LINC
00507', '57': 'LINC00508', '61': 'NPTX1', '64': 'KCNIP1', '60': 'TBR1', '6
2': 'LINC00710', '58': 'ANKRD33B', '59': 'LINC00152'}, 'Importance': {'53':
0.09941479067973553, '55': 0.09725362729681845, '54': 0.07171989115283424,
'57': 0.05383117013159357, '61': 0.048527845718618325, '64': 0.0433285139373
93155, '60': 0.038116946621593664, '62': 0.03457181494856369, '58': 0.032358
086581950704, '59': 0.03127609453692296}}}
```

In [18]:

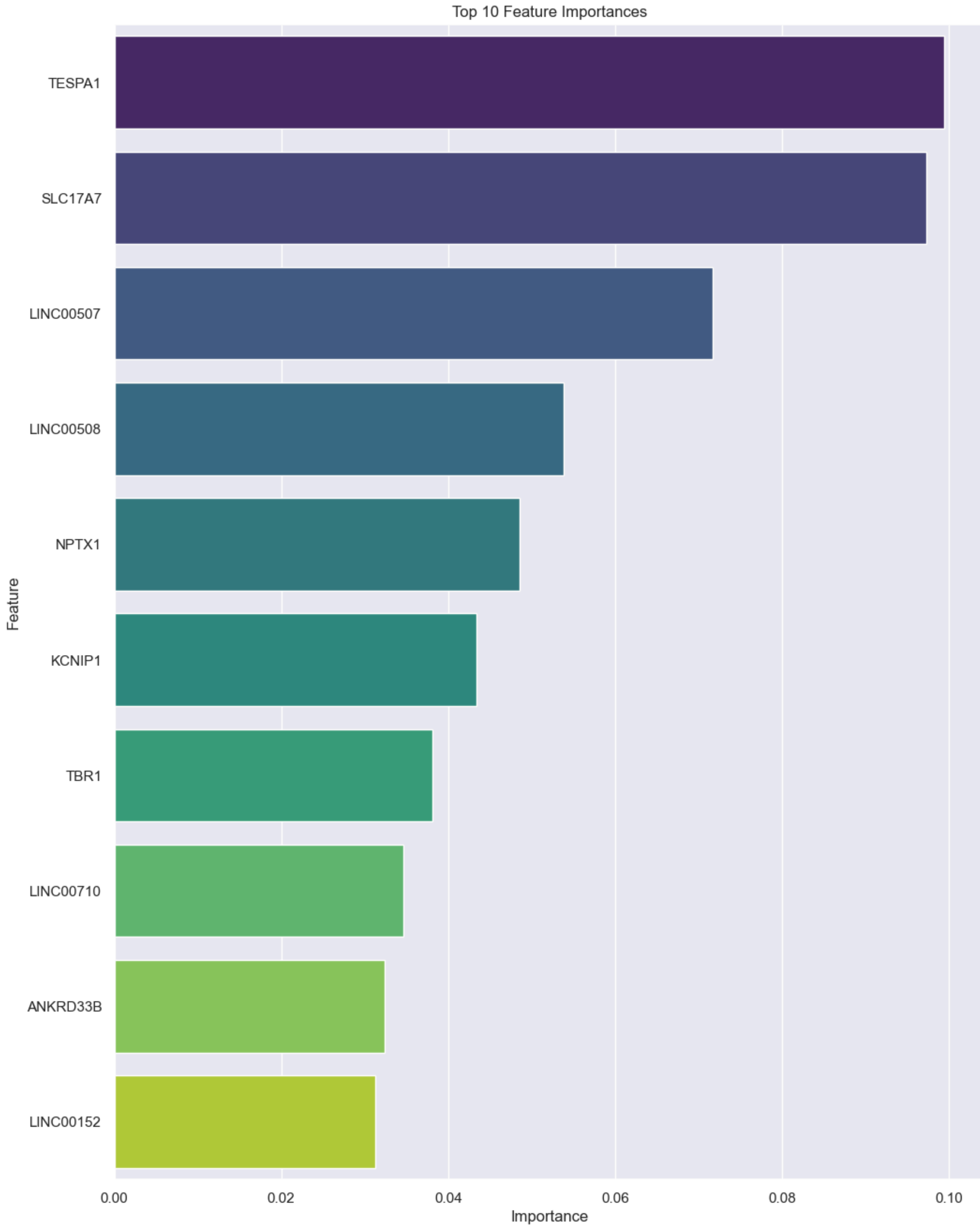
```
# Just consider the oob,f1, cv score
# Initialize variables to store the best key and its metrics
best_key = None
best_metrics = {'OOB Score': 0, 'Mean cross-validation score': 0, 'F1 Score'

# Iterate through each key in the JSON data
for key, metrics in results.items():
    # Compare the current key's metrics with the best metrics
    confusion_matrix = metrics["Confusion_matrix"]
    f1_score = calculate_f1_score(confusion_matrix)
    if (metrics['OOB Score'] > best_metrics['OOB Score'] and
        metrics['Mean cross-validation score'] > best_metrics['Mean cross-va
        f1_score > best_metrics['F1 Score']):
        best_key = key
        best_metrics['OOB Score'] = metrics['OOB Score']
        best_metrics['Mean cross-validation score'] = metrics['Mean cross-va
        best_metrics['F1 Score'] = f1_score

# Print the best key and its metrics
print(f"Best Key: {best_key}")
print(f"Metrics: {best_metrics}")
```

```
Best Key: Ntree:500, Mtry:50
Metrics: {'OOB Score': 0.992816091954023, 'Mean cross-validation score': 0.9
931152980210926, 'F1 Score': 1.0}
```

In [19]:
```
# plot the best results
conf_matrix = best_metrics['Confusion_matrix']
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='ocean_r')
# Add labels and title
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
# Display the heatmap
plt.show()
print(conf_matrix)

# Visualize the top 10 feature importances
top_10_features = pd.DataFrame(best_metrics['Top 10 Features'])
plt.figure(figsize=(12, 16))
sns.barplot(x='Importance', y='Feature', data=top_10_features, palette='viri
plt.title('Top 10 Feature Importances')
plt.show()
```




```
[[109, 0], [0, 66]]
```



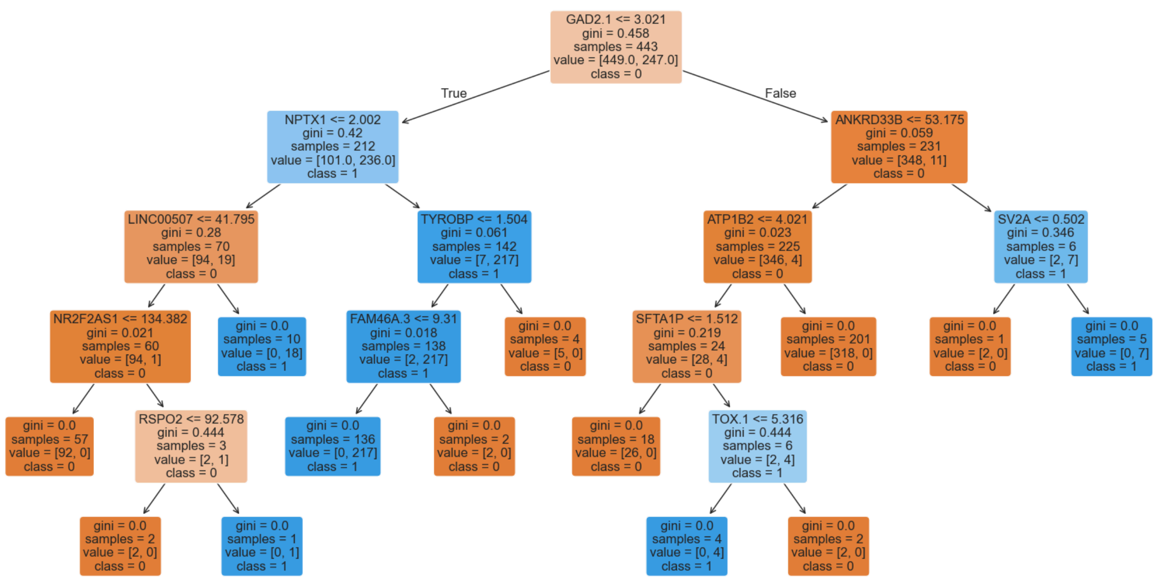
In [18]:
```
# Retrain the model with the best hyperparameters
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
# Load the dataset and split it
raw_data = pd.read_csv('datasets/e1 positive.csv')
y = raw_data['Label']
x = raw_data.drop('Label', axis=1)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rar

model = RandomForestClassifier(n_estimators=500, max_features=50, oob_score=
```

```
model.fit(x_train, y_train)

# Find one example each for label 1 and label 0
example_1 = x_test[y_test == 1].iloc[0]
example_0 = x_test[y_test == 0].iloc[0]
p_pre = model.predict_proba([example_1, example_0])
print(p_pre)
# Visualize the decision process for the example with label 1
plt.figure(figsize=(20, 10))
plot_tree(model.estimators_[0], feature_names=x.columns, filled=True, rounde
plt.title('Decision Tree for Example with Label 1')
plt.show()

# Visualize the decision process for the example with label 0
plt.figure(figsize=(20, 10))
plot_tree(model.estimators_[1], feature_names=x.columns, filled=True, rounde
plt.title('Decision Tree for Example with Label 0')
plt.show()
```

```
[[0.004 0.996]
 [0.934 0.066]]
```

```
/opt/anaconda3/envs/myFirstEnv/lib/python3.12/site-packages/sklearn/base.py:
493: UserWarning: X does not have valid feature names, but RandomForestClass
ifier was fitted with feature names
  warnings.warn(
```

Decision Tree for Example with Label 1



Decision Tree for Example with Label 0