Python Coding Mission I

Python Coding Mission I : Random Forest

1. Breast Cancer Dataset (scikit-learn) – Data Summary & Preprocessing

- **Load data & check summary**
  - Understand dataset size: total samples & number of features
  - Reference: 1658 LUAD patients, 724 cancer patients in the study

- **Check target label distribution (positive/negative)**
  - Identify class imbalance
  - Reference: High-risk vs. low-risk groups analyzed for survival, immune response, clinical traits; validated via KM curves & ROC

- **Check for missing values**
  - Prevent errors and performance degradation during training4
  - Reference: Performed preprocessing like normalization and removing missing genes

- **Examine statistics of mean features**
  - Identify features with high discriminative power
  - Reference: Selected 7 key genes using LASSO-Cox and CoxBoost models

Python Coding Mission I : Random Forest

```python
# Breast Cancer 데이터 분석 및 모델링 실습

from collections import defaultdict

import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score,
    auc,
    classification_report,
    confusion_matrix,
    roc_curve,
)
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, train_test_split
```

- **Built-in Data Structures / Container Modules**
  `defaultdict`
- **Model Evaluation / Metric Functions**
  `accuracy_score`,
  `confusion_matrix`,
  `roc_curve`,
  `auc`, `classification_report`
- **Hyperparameter Tuning Tools**
  `GridSearchCV`
  `RandomizedSearchCV`

Python Coding Mission I : Random Forest

1. Load and summarize the Breast Cancer dataset from scikit-learn

(a) total samples and features

```python
# 1. 데이터 로딩 및 요약 분석
cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
feature_names = cancer.feature_names
num_features = len(feature_names)

# 1-a. 샘플 수 및 특성 수 확인
print("전체 샘플 수:", X.shape[0])
print("특성 수:", X.shape[1])
```

```
전체 샘플 수: 569
특성 수: 30
```

- ***load_breast_cancer()*** : Load the Breast Cancer dataset
- **X** = cancer.data : Numerical feature values of tumors (samples × features)
- **y** = cancer.target : Diagnosis result (0 = malignant, 1 = benign)
- ***.feature_names*** : List of feature names

Use NumPy functions
`X.shape` → (number of samples, number of features)
`X.shape[0]` → Number of rows = number of samples
`X.shape[1]` → Number of columns = number of features

```python
data = cancer.data
num_sample = len(data)
num_feature = len(data[0])
print(f"전체 샘플 수: {num_samples}")
print(f"features 수: {num_features}")
```

- `data`: list of lists
- `len(data)` → total number of samples (rows)
- `len(data[0])` → number of features per sample (columns)

Python Coding Mission I : Random Forest

1. Load and summarize the Breast Cancer dataset from scikit-learn

(b) Distribution of target labels (benign/malignant)

```
# 1-b. 라벨 분포 확인 (0: malignant, 1: benign)
target = load_breast_cancer().target
counts = {0: 0, 1: 0}
for t in target:
    counts[t] += 1

print(f"malignant (0) 샘플 개수: {counts[0]}")
print(f"benign (1) 샘플 개수: {counts[1]}")
```

```
malignant (0) 샘플 개수 : 212
benign (1) 샘플 개수 : 357
```

- ***counts={0:0, 1:0}*** **:** Dictionary for counting samples per label
- ***for t in target*** : Iterate through target labels to count samples of malignant and benign tumors

If t = **0** → counts[0] += 1 (**malignant** count)
If t = **1** → counts[1] += 1 (**benign** count)

→ **The target distribution is fairly balanced**

→ **Background for potentially high AUC (1)**

Python Coding Mission I : Random Forest

1. Load and summarize the Breast Cancer dataset from scikit-learn

(c) Check if there are any features with missing values

```
# 1-c. 결측값 확인
missing_count = 0
for row in cancer.data:
    for x in row:
        if x is None:
            found_missing = True
            missing_count += 1

if missing_count != 0:
    print("결측값 존재")
else:
    print("결측값 없음")
```

```
결측값 없음
```

- ***missing_count = 0***
  Variable to count missing values → initialize to 0
- ***for row in cancer.data:***
  Iterate through the entire dataset and check if each value x is **None**
- ***if x is None:***
  `missing_count` += 1 (increment the count)

```
if np.isnan(X).any():
    print("결측값 존재")
else:
    print("결측값 없음")
```

Use NumPy functions
- `np.isnan(X)` : returns True/False for each element in array X indicating if it is NaN
- `.any()` : returns True if at least one element is True

Python Coding Mission I : Random Forest

1. Load and summarize the Breast Cancer dataset from scikit-learn

(d) Check descriptive statistics for mean-related features and summarize key characteristics.

```
# 1-d. mean 관련 feature 요약
radius_idx = list(feature_names).index("mean radius")

radius = []
for row in X:
    radius.append(row[radius_idx])
mean_radius = sum(radius) / len(radius)
min_radius = min(radius)
max_radius = max(radius)

print("mean radius 평균:", mean_radius)
print("mean radius 최소값:", min_radius)
print("mean radius 최대값:", max_radius)
```

```
mean radius 평균 : 14.127291739894552
mean radius 최소값 : 6.981
mean radius 최대값 : 28.11
```

- **radius_idx**
  Find the column index corresponding to the name "mean radius“
- **for row in X**
  Extract the value corresponding to `radius_idx` (mean radius) from each row and store it in the list

```
radius = X[:, radius_idx]
mean_radius = np.mean(radius)
min_radius = np.min(radius)
max_radius = np.max(radius)
```

Use NumPy functions

- radius = X[:, radius_idx]:
  Extract only the radius_idx-th column (mean radius) from all rows of X

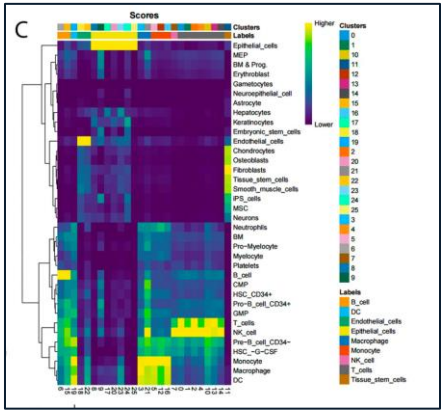
Python Coding Mission I : Random Forest

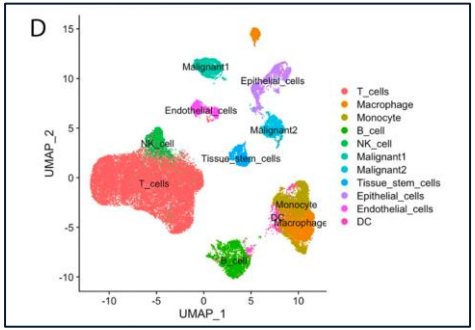
2. Data Visualization

a) **Histogram:** Distribution of mean radius and mean texture

b) **Scatter plot:** mean radius vs. mean texture

c) **Box plot:** Distribution of mean area by target label





<Reference>

Visualize the distribution and characteristics of specific cell groups using UMAP plots, heatmaps, etc

Python Coding Mission I : Random Forest

2. Data Visualization

```
# 2. 데이터 시각화
texture_idx = list(feature_names).index("mean texture")
texture = X[:, texture_idx]
```

(a) Visualize the distribution of mean radius and mean texture using histograms.

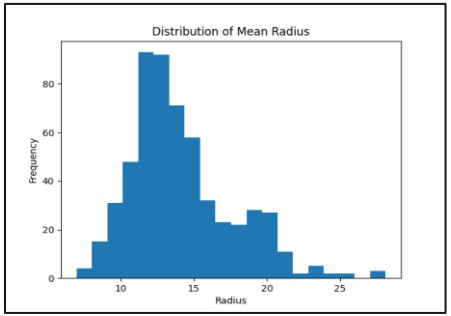
```
# 2. 데이터 시각화
texture_idx = list(feature_names).index("mean texture")
texture = []
for row in X:
    texture.append(row[texture_idx])
```

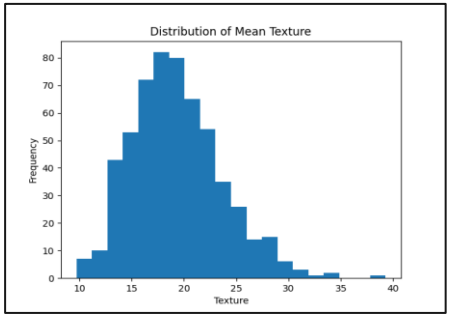
```
# 2-a. 히스토그램 시각화
plt.hist(radius, bins=20)
plt.title("Distribution of Mean Radius")
plt.xlabel("Radius")
plt.ylabel("Frequency")
plt.show()
```

```
plt.hist(texture, bins=20)
plt.title("Distribution of Mean Texture")
plt.xlabel("Texture")
plt.ylabel("Frequency")
plt.show()
```

- ***texture*** : List of mean texture values from all samples
- ***plt.hist()*** : Function to plot a histogram
- ***bins=20***: Divide the data into 20 bins

  -> check where the data is concentrated and identify any outliers



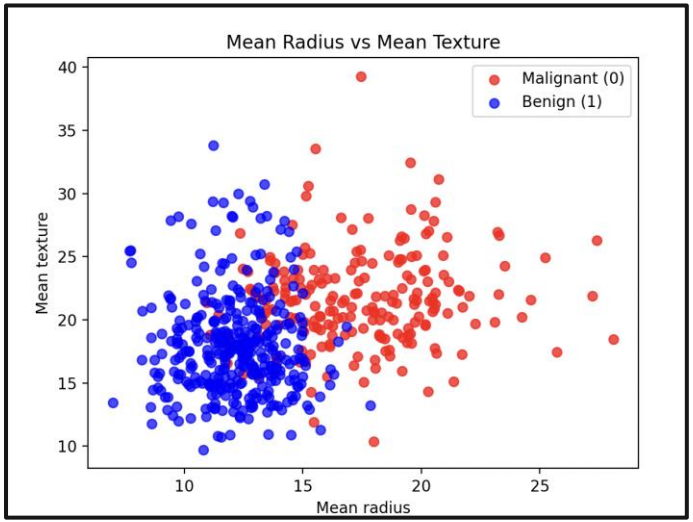


Python Coding Mission I

2. Data Visualization

(b) **Plot a scatter plot of mean radius vs. mean texture, coloring points differently by target label.**

```python
# 2-b. 산점도 (target별 색상)
plt.scatter(
    radius[y == 0], texture[y == 0], color="red", alpha=0.7, label="Malignant (0)"
)
plt.scatter(
    radius[y == 1], texture[y == 1], color="blue", alpha=0.7, label="Benign (1)"
)
plt.title("Mean Radius vs Mean Texture")
plt.xlabel("Mean radius")
plt.ylabel("Mean texture")
plt.legend(loc="upper right")
plt.show()
```



- ***plt.scatter()*** : Each point represents a sample-> scatter plot of **malignant(0) benign(1)**
- ***alpha=0.7*** : Adjust transparency so overlapping points remain visible

If classes are clearly separated → low noise and well-structured variables

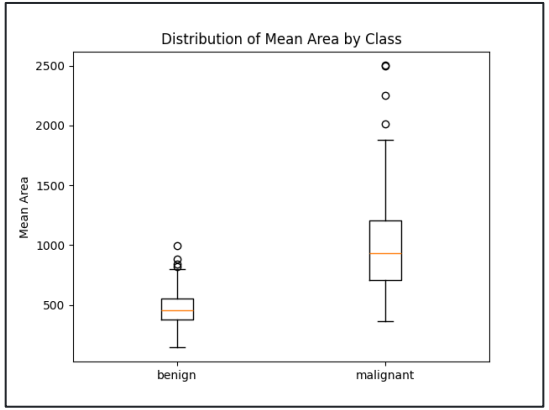
➜ **Background for potentially high AUC (2)**

Python Coding Mission I

2. Data Visualization

(c) Visualize the distribution of mean area by target label using a boxplot.

```
# 2-c. mean area의 라벨별 분포 (박스플롯)
area_idx = list(feature_names).index("mean area")
area_positive = []  # target == 1 (benign)
area_negative = []  # target == 0 (malignant)
for row, t in zip(X, y):
    if t == 1:
        area_positive.append(row[area_idx])
    else:
        area_negative.append(row[area_idx])

plt.boxplot([area_positive, area_negative], labels=["benign", "malignant"])
plt.title("Distribution of Mean Area by Class")
plt.ylabel("Mean Area")
plt.show()
```



- **Mean tumor area**: Larger values and greater variance observed in the malignant class

```
# 2-c. mean area의 라벨별 분포 (박스플롯)
area_idx = list(feature_names).index("mean area")
area = X[:, area_idx]
plt.boxplot([area[y == 1], area[y == 0]], labels=["benign", "malignant"])
plt.title("Distribution of Mean Area by Class")
plt.ylabel("Mean Area")
plt.show()
```

- for row, t in zip(x, y): For each sample (row) and label (t),
  append the sample to the corresponding list based on its label
- -> Compare the two lists using ***plt.boxplot() :***to visualize differences in feature distributions

Python Coding Mission I : Random Forest

3. Machine Learning Data Preparation

(a) Split data into **training** and **test** sets for modeling
Test size: 20%
Random state: 77 (fixed for reproducibility)

X y

```python
# 3. 학습/테스트 데이터의 분할
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=77)

print("train:", len(X_train))
print("test:", len(X_test))
```

```
train: 455
test: 114
```

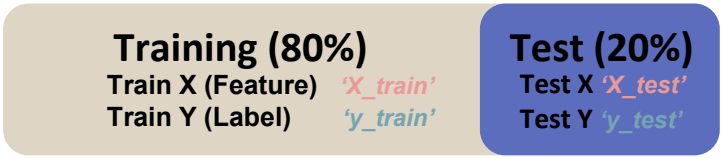
<Reference> Based on TCGA LUAD Bulk RNA-seq data, a 7-gene signature was constructed, and the same model evaluation process was used to validate its predictive performance.

scRNA seq » **Bulk RNA seq**

Python Coding Mission I : Random Forest

4. Model Creation & Test Evaluation

(a) Train a classification model (Base model) on the training data using the RandomForestClassifier **class** and perform predictions.



➜ **Functions from the scikit-learn package**

- ***fit()***: Train the model using the training data
- ***predict()***: Use the trained model to predict class labels on the test data -> **Malignant(0)**, **Benign(1)**
- ***predict_proba()***: Return the predicted probabilities for each class on the test data -> ***[:, 1]***: Probability of the positive class

```python
# 4. 기본 모델 학습 및 평가
from sklearn.ensemble import RandomForestClassifier

base_clf = RandomForestClassifier(random_state=77)
base_clf.fit(X_train, y_train)
y_pred = base_clf.predict(X_test)
y_proba = base_clf.predict_proba(X_test)[:, 1]
```

```
y_pred:[1 0 1 0 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 1 1 1 0 1 1 0
 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1
 1 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1]
y_proba:[0.99 0.   1.   0.   0.01 0.03 0.97 0.94 0.65 0.   0.   0.9  1.   1.
 1.   1.   1.   0.97 0.99 1.   0.   0.41 0.   0.19 0.98 0.95 1.   1.
 1.   0.05 1.   0.53 1.   0.04 0.99 0.95 0.   0.01 0.   0.97 1.   0.
 0.01 0.78 0.97 0.99 0.02 0.15 0.98 1.   0.7  0.   0.99 0.25 0.93 0.98
 1.   0.6  0.   1.   1.   0.99 0.51 0.99 0.99 0.   0.11 0.9  0.1  0.23
 1.   0.75 0.61 0.84 0.99 0.   0.07 0.13 1.   0.99 1.   0.32 0.99 0.92
 1.   0.03 1.   1.   0.99 0.11 0.99 0.   0.   1.   1.   1.   1.   0.99
 0.97 0.04 0.94 0.97 1.   0.91 1.   0.   0.98 0.96 0.94 1.   1.   0.96
 0.98 0.79]
```

Python Coding Mission I : Random Forest

4. Model Creation & Test Evaluation

(b) Make predictions on the **test dataset** and **Evaluate** using:
>> **ROC curve / AUC / Accuracy/ Confusion Matrix / Classification Report**

→ Func

ROC curve

```
import matplotlib.pyplot as plt
from sklearn.metrics import (
    accuracy_score,
    auc,
    classification_report,
    confusion_matrix,
    roc_curve,
)

# ROC곡선 좌표생성
fpr, tpr, _ = roc_curve(y_test, y_proba)
print(fpr)
print(tpr)

# 좌표값 기준 AUC값 도출
roc_auc = auc(fpr, tpr)
print("AUC:", roc_auc)
```

```
AUC: 0.9823931623931624
```

Accuracy 관련 지표

```
acc = accuracy_score(y_test, y_pred) # 정확도
cm = confusion_matrix(y_test, y_pred) # 혼동행렬
clf_report = classification_report(y_test, y_pred) # 분류리포트

print("정확도:", acc)
print("혼동 행렬:\n", cm)
print("분류 리포트:\n", clf_report)
```

```
정확도: 0.9385964912280702
혼동 행렬:
 [[34  5]
 [ 2 73]]
분류 리포트:
              precision    recall  f1-score   support

           0       0.94      0.87      0.91        39
           1       0.94      0.97      0.95        75

    accuracy                           0.94       114
   macro avg       0.94      0.92      0.93       114
weighted avg       0.94      0.94      0.94       114
```

TN (True Negative) FP (False Positive)
FN (False Negative) TP (True Positive)

Python Coding Mission I : Random Forest

4. Model Creation & Test Evaluation

(b) Make predictions on the **test dataset** and **Evaluate** using:
>> **ROC curve / AUC** / **Accuracy/ Confusion Matrix / Classification Report**

ROC curve

→ Func

```
import matplotlib.pyplot as plt
from sklearn.metrics import (
    accuracy_score,
    auc,
    classification_report,
    confusion_matrix,
    roc_curve,
)

# ROC곡선 좌표생성
fpr, tpr, _ = roc_curve(y_test, y_proba)
print(fpr)
print(tpr)

# 좌표값 기준 AUC값 도출
roc_auc = auc(fpr, tpr)
print("AUC:", roc_auc)
```

```
AUC: 0.9823931623931624
```

Accuracy 관련 지표

```
acc = accuracy_score(y_test, y_pred) # 정확도
cm = confusion_matrix(y_test, y_pred) # 혼동행렬
```

❶ fpr

```
[0.         0.         0.02564103 0.02564103 0.02564103 0.02564103
 0.02564103 0.02564103 0.02564103 0.02564103 0.05128205 0.05128205
 0.12820513 0.12820513 0.25641026 0.30769231 0.38461538 0.48717949
 0.51282051 0.58974359 1.        ]
```

```
정확도: 0.9385964912280702
혼동 행렬:
 [[34  5]
```

TN (True Negative) FP (False Positive)

❷ tpr

```
[0.         0.41333333 0.58666667 0.65333333 0.73333333 0.78666667
 0.82666667 0.86666667 0.89333333 0.93333333 0.93333333 0.96
 0.96       1.         1.         1.         1.         1.
 1.         1.         1.        ]
```

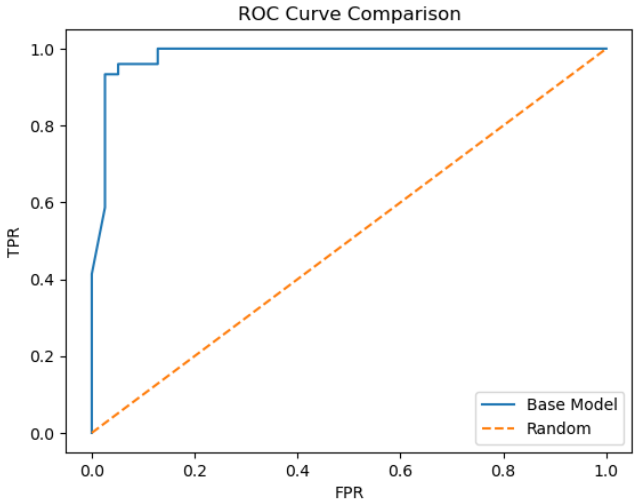
```
              accuracy                           0.94       114
             macro avg       0.94      0.92      0.93       114
          weighted avg       0.94      0.94      0.94       114
```

Python Coding Mission I : Random Forest

4. Model Creation & Test Evaluation

(b) Make predictions on the **test dataset** and **Evaluate** using:
>> **ROC curve / AUC / Accuracy/ Confusion Matrix / Classification Report**

ROC curve




```
AUC: 0.9823931623931624
```

Accuracy 관련 지표

```python
acc = accuracy_score(y_test, y_pred) # 정확도
cm = confusion_matrix(y_test, y_pred) # 혼동행렬
clf_report = classification_report(y_test, y_pred) # 분류리포트

print("정확도:", acc)
print("혼동 행렬:\n", cm)
print("분류 리포트:\n", clf_report)
```

```
정확도: 0.9385964912280702
혼동 행렬:
 [[34  5]
 [ 2 73]]
분류 리포트:
              precision    recall  f1-score   support

           0       0.94      0.87      0.91        39
           1       0.94      0.97      0.95        75

    accuracy                           0.94       114
   macro avg       0.94      0.92      0.93       114
weighted avg       0.94      0.94      0.94       114
```

TN (True Negative) FP (False Positive)
FN (False Negative) TP (True Positive)

Python Coding Mission I : Random Forest

5. Hyperparameter Tuning

(a) Optimize model hyperparameters using **GridSearchCV** or **RandomizedSearchCV**

RandomizedSearchCV « GridSearchCV

Randomly sample a subset of hyperparameter combinations for exploration

Exhaustively explore **all** hyperparameter combinations(🕐🔺)

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# 5. 하이퍼파라미터를 최적화 by GridSearchCV
param_grid = {
    "n_estimators": [1, 50, 100, 200],
    "max_depth": [1, 50, 100, 200],
}
grid = GridSearchCV(
    estimator=base_clf,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
)
grid.fit(X_train, y_train)
```

`param_grid`: Hyperparameter Range Setting
- ***n_estimators***: number of trees
- ***max_depth***: maximum depth of each decision tree

`GridSearchCV`
- ***cv***: alternate training/validation to evaluate generalization performance🔺
- ***scoring***: performance metric for evaluation

Python Coding Mission I : Random Forest

5. Hyperparameter Tuning

(a) Optimize model hyperparameters using **GridSearchCV** or **RandomizedSearchCV**

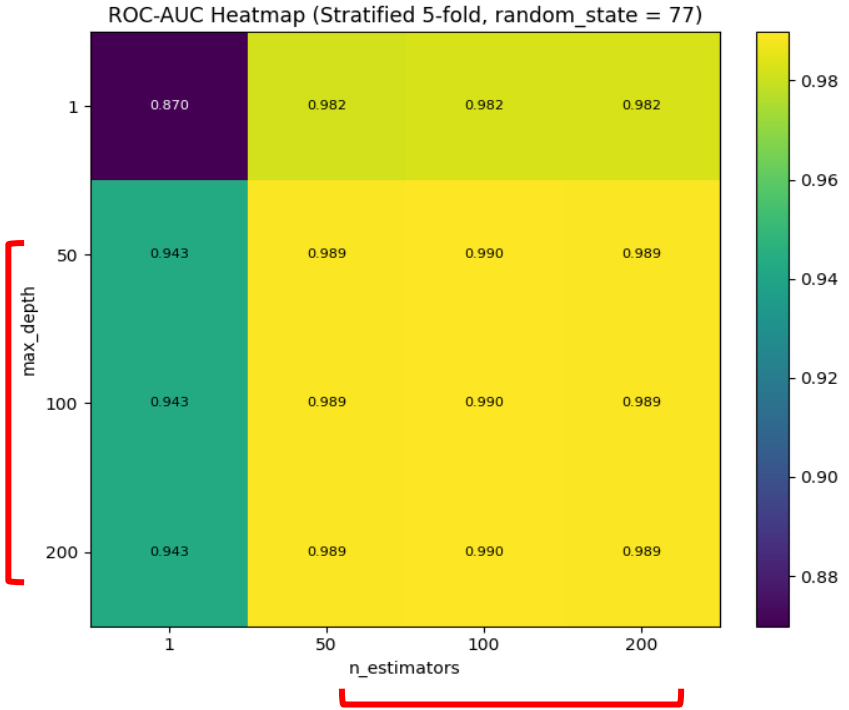
GridSearchCV

Set parameter ranges for optimal combination

- `n_estimators` (default: 100) ➜ [10, 50, 100, 200]
- `max_depth` (default: None) ➜ [1, 50, 100, 200]

For Potential Overfitting ▼

<Reference> Hyperparameter optimization was performed using GridSearch during the model tuning process.



Python Coding Mission I : Random Forest

5. Hyperparameter Tuning

(b) Display the **optimal parameters** and corresponding **cross-validation accuracy**

➜ grid.fit 의 method

- ***grid.best_params_*, *grid.best_score_***
  : Built-in attributes of a **GridSearchCV** object
  Automatically generated and stored after executing **gird.fit**

```
최적 하이퍼파라미터 (Grid): {'max_depth': 50, 'n_estimators': 200}
최고 CV 정확도 (Grid): 0.9648351648351647
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# 5. 하이퍼파라미터 튜닝 (GridSearchCV)
param_grid = {
    "n_estimators": [1, 50, 100, 200],
    "max_depth": [1, 50, 100, 200],
}
grid = GridSearchCV(
    estimator=RandomForestClassifier(random_state=77),
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1,
    verbose=1,
)
grid.fit(X_train, y_train)
print("최적 하이퍼파라미터 (Grid):", grid.best_params_)
print("최고 CV 정확도 (Grid):", grid.best_score_)
```

Python Coding Mission I : Random Forest

6. Final Model Performance Evaluation

(a) Compare model performance before and after hyperparameter tuning

Metrics: Accuracy, Confusion Matrix, Classification Report

```python
from sklearn.metrics import
(accuracy_score, classification_report, confusion_matrix)

# 6. 튜닝 전/후 모델 성능 비교
print("=== 튜닝 전 모델 ===")
y_pred_base = base_clf.predict(X_test)
print("정확도:", accuracy_score(y_test, y_pred_base))
print("혼동 행렬:\n", confusion_matrix(y_test, y_pred_base))
print("분류 리포트:\n", classification_report(y_test, y_pred_base))

print("=== 튜닝 후 모델 (GridSearchCV 기준) ===")
best_clf = grid.best_estimator_
y_pred_best = best_clf.predict(X_test)
print("정확도:", accuracy_score(y_test, y_pred_best))
print("혼동 행렬:\n", confusion_matrix(y_test, y_pred_best))
print("분류 리포트:\n", classification_report(y_test, y_pred_best))
```

```
=== 튜닝 전 모델 ===
정확도: 0.9385964912280702
혼동 행렬:
 [[34  5]
 [ 2 73]]
```

TN (True Negative) FP (False Positive)

FN (False Negative) TP (True Positive)

분류 리포트:

	precision	recall	f1-score	support
0	0.94	0.87	0.91	39
1	0.94	0.97	0.95	75
accuracy			0.94	114
macro avg	0.94	0.92	0.93	114
weighted avg	0.94	0.94	0.94	114

```
=== 튜닝 후 모델 (GridSearchCV 기준) ===
정확도: 0.9473684210526315
혼동 행렬:
 [[34  5]
 [ 1 74]]
```

TN (True Negative) FP (False Positive)

FN (False Negative) TP (True Positive)

분류 리포트:

	precision	recall	f1-score	support
0	0.97	0.87	0.92	39
1	0.94	0.99	0.96	75
accuracy			0.95	114
macro avg	0.95	0.93	0.94	114
weighted avg	0.95	0.95	0.95	114

Before tuning → **GridSearch** → **After tuning**

Python Coding Mission I : Random Forest

7. ROC Curve Visualization

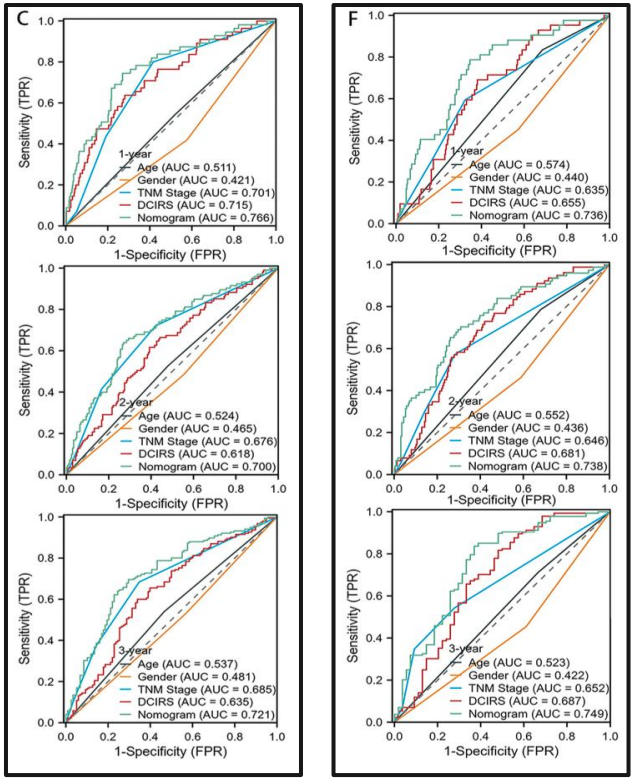
a. Plot ROC curves for both the baseline and tuned models on the same graph

b. Display and compare the **AUC (Area Under Curve)** for each model

<Reference> Fig. 3C, F – ROC Curves of Nomograms (TCGA & GSE72094)

- Visual evaluation of model predictive performance
- Overlay ROC curves of **baseline** and **hyperparameter-tuned** models in a single plot
- Compare **AUC values** to assess performance improvement of Group 4's model

.



Python Coding Mission I : Random Forest

7. ROC Curve Visualization

a. Plot ROC curves for both the baseline and tuned models on the same graph

b. Display and compare the **AUC (Area Under Curve)** for each model

```python
import matplotlib.pyplot as pl
from sklearn.metrics importt(auc, roc_curve)

# 7. ROC curve 시각화
fpr_base, tpr_base, _ = roc_curve(y_test, base_clf.predict_proba(X_test)[:, 1])
auc_base = auc(fpr_base, tpr_base)

fpr_tuned, tpr_tuned, _ = roc_curve(y_test, best_clf.predict_proba(X_test)[:, 1])
auc_tuned = auc(fpr_tuned, tpr_tuned)

plt.plot(
    *roc_curve(y_test, base_clf.predict_proba(X_test)[:, 1])[:2],
    label=f"Base Model (AUC = {auc_base:.3f})",
)
plt.plot(
    *roc_curve(y_test, best_clf.predict_proba(X_test)[:, 1])[:2],
    label=f"Tuned Model (AUC = {auc_tuned:.3f})",
)
plt.plot([0, 1], [0, 1], "--", label="Random")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("7-a,b. ROC Curve Comparison")
plt.legend(loc="lower right")
plt.show()
```

➜ ROC curve method

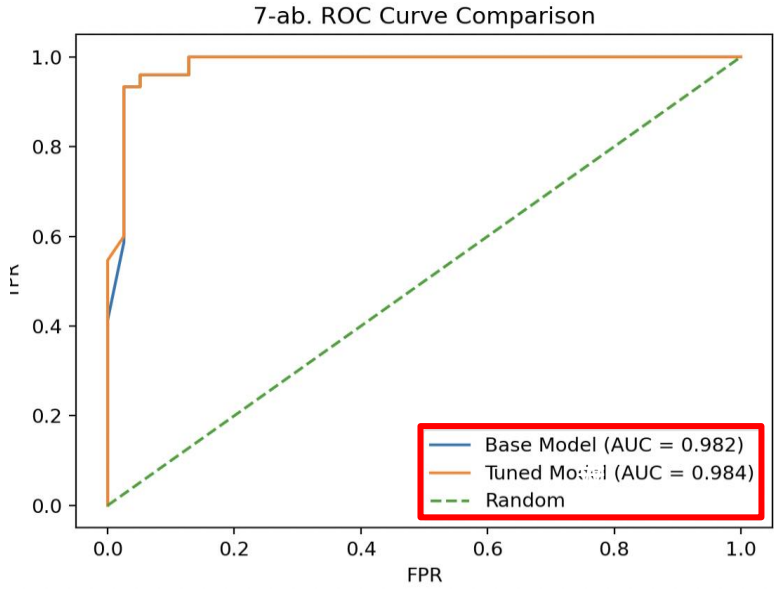
- ***fpr_base, tpr_base, _ = roc_curve(y_test, base_clf.predict_proba(X_test)[:, 1])***
  : fpr_base (X-axis values of the ROC curve)
  : tpr_base (Y-axis values of the ROC curve)

- ***auc_base = auc(fpr_base, tpr_base)***
  : Calculate AUC using false positive rate (FPR) and true positive rate (TPR)

- ***plt.plot(...)*** **:** Plot the ROC curve using and TPR values obtained from the roc_curve function.

(Slight/minor difference)

Python Coding Mission I : Random Forest



Base Model = 0.982 / Tuned Model = 0.984
Difference : 0.002 (minor difference)

Python Coding Mission I : Random Forest

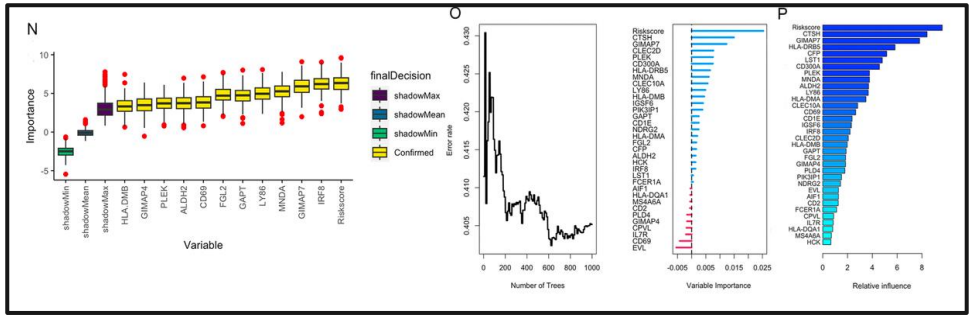
8. Feature Selection

a. Extract feature importance scores

b. Visualize the top 10 features using a horizontal bar chart

X-axis: Importance

Y-axis: Feature names



<Reference> Constructed DCIRS and identified 7 core genes

Identifying 7 core genes = selecting the top 10 most important features

These 7 final core genes serve as the **key features** in the model

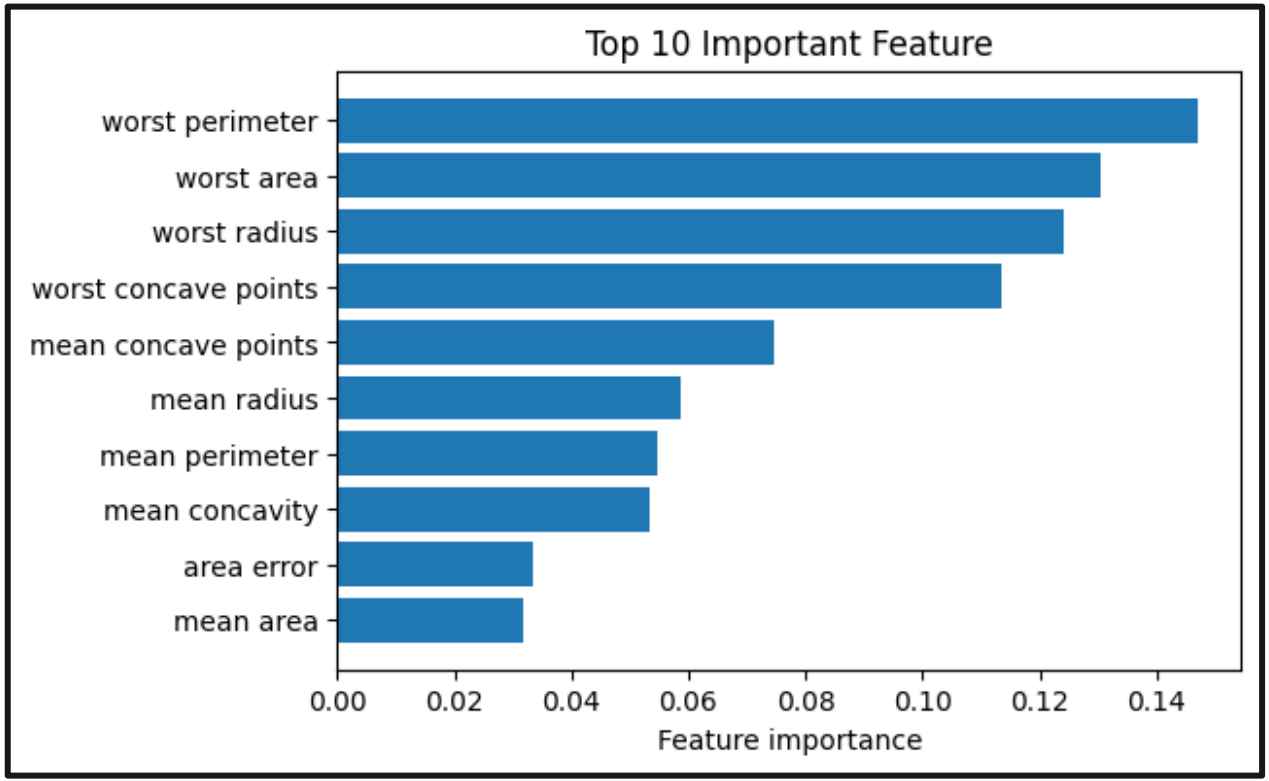
Python Coding Mission I : Random Forest

```python
# 8-a. feature의 중요도를 추출하세요
importances = best_clf.feature_importances_  # 숫자들
names = list(feature_names)  # 예: ["mean radius", "mean texture", ...]
feat_imp_pairs = list(zip(names, importances))
feat_imp_pairs.sort(key=lambda x: x[1], reverse=True)
top10 = feat_imp_pairs[:10]
labels = [name for name, imp in top10]
values = [imp for name, imp in top10]

# 8-b. 상위 10개의 중요 feature를 수평 막대 그래프로 시각화하세요 (x축: 중요도 / y축: feature 이름)
# 4) 수평 막대그래프로 시각화
plt.figure(figsize=(6, 4))
plt.barh(labels[::-1], values[::-1])  # 뒤집어서 가장 중요한 게 위에 오도록
plt.xlabel("Feature importance")
plt.title("Top 10 Important Feature")
plt.show()
```

Feature Importances method

- ***.feature_importances_*** : Return the importance scores of each input feature as an array
- ***feat_imp_pairs.sort(key=lambda x: x[1], reverse=True)*** : Sort in descending order based on importance (x[1])
- ***top10 = feat_imp_pairs[:10]*** : Extract the top 10 most important features from the sorted feature list
- ***plt.barh(labels[::-1], values[::-1])*** : Plot a horizontal bar chart of the top 10 features by importance
  [::-1] ensures the most important feature appears at the top.

Python Coding Mission I : Random Forest



1. Worst Radius
: Maximum radius of the tumor nucleus

2. Worst Area
: Maximum area of the tumor nucleus

3. Worst Radius
: Maximum perimeter of the tumor nucleus

4. **Worst Concave Points**
: Maximum number of concave points of the nucleus

5. **Mean Concave Points**
: Average number of concave points

6. **Mean Radius**
: Average radius of the nucleus

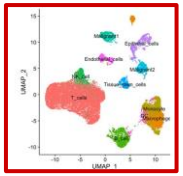
7. **Mean Perimeter**
: Average perimeter of the nucleus

8. **Mean Concavity**
: Average concavity of the nucleus

9. **Area error**
: Standard error of the nucleus area

10. **Mean area**
Average area of the nucleus

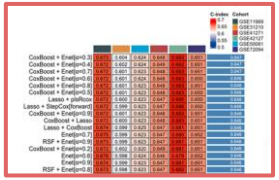
Paper Flow



Gene set selection

scRNA seq

- Identification of Candidate DC Marker Genes (83 Genes) from scRNA-seq Data

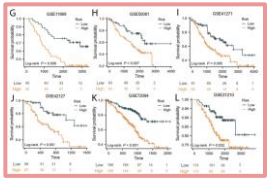


Construction of DCIRS (Machine Learning)

bulk RNA seq

CoxBoost+ENET

- Calculation of DCIRS (Dendritic Cell Implicated Risk Score) Comprising 7 Genes Using Machine Learning Based on Bulk RNA-seq Data (6 Public Datasets)

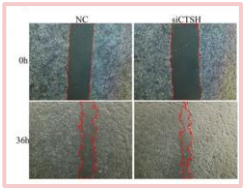


<Validation>
1. Survival analysis
2. Drug Response

Kaplan Meier curve

IC50 response

- Validation of DCIRS Prognostic Power in the Dataset Using Survival Analysis and ROC Curves
- Comparison of Targeted Therapy and Chemotherapy (IC50) Drug Sensitivity According to DCIRS Levels



New gene target finding

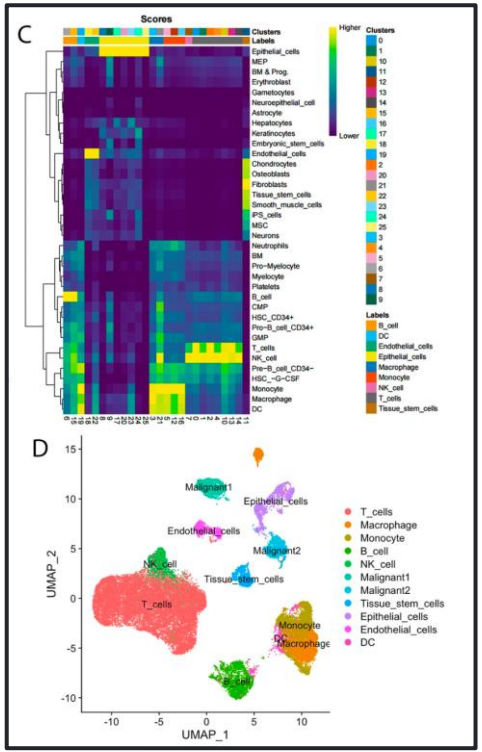
In vitro assay

- Validation of CTSH Tumor-Suppressive Function (EMT Inhibition, Reduced Cell Migration/Invasion, Induced Apoptosis) via In Vitro Experiments (qRT-PCR, Western Blot, Wound Healing, Transwell, Apoptosis Assays, etc.)

Introduction

Machine-learning and combined analysis of single-cell and bulk-RNA sequencing identified a DC gene signature to predict prognosis and immunotherapy response for patients with lung adenocarcinoma

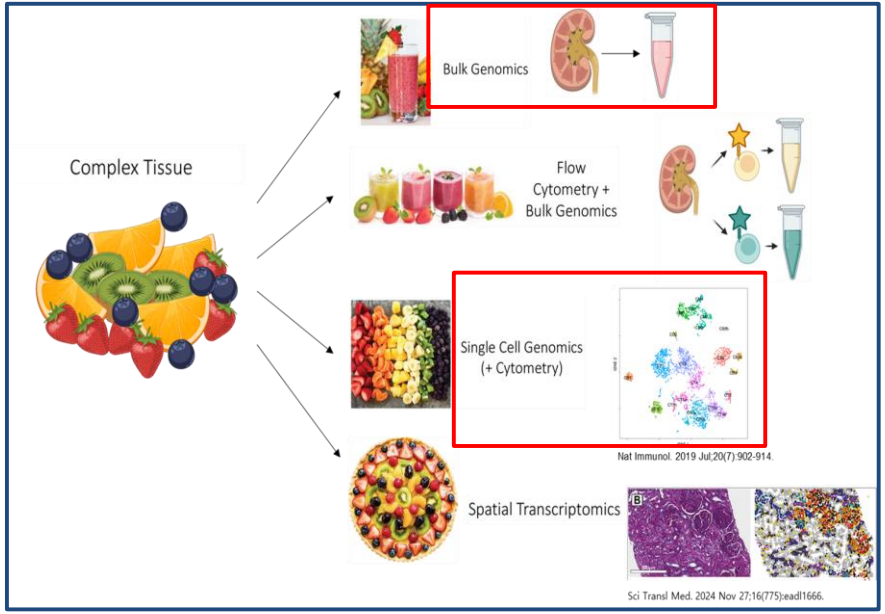
KEY Hypothesis



- Dendritic cell (DC)-related **gene signatures** can serve as a strong and effective **predictive biomarker** for prognosis and immunotherapy response in lung adenocarcinoma patients.

- Among the genes comprising the DC gene signature, CTSH is expected to act as a **novel protective factor in LUAD**, exerting **tumor-suppressive effects**.

Bulk RNA-seq & Single-cell RNA-seq



- **Bulk RNA-seq**

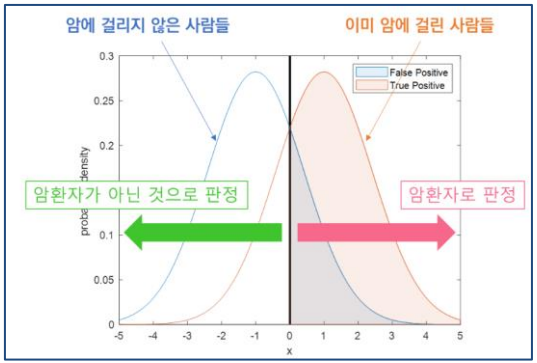
- Sequencing the total RNA extracted from a large population of cells to measure gene expression levels
- Provides an average gene expression profile across all cells in the sample, rather than characteristics of individual cells

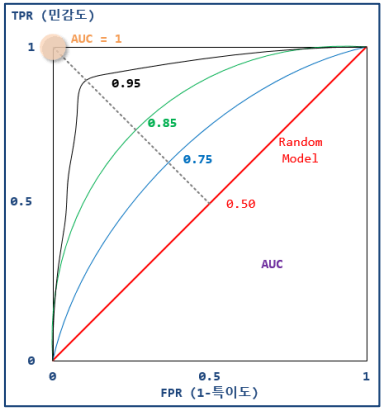
- **Single-cell RNA-seq**

- RNA is extracted and analyzed from individual cells.
- Reveals cell-to-cell differences, changes in cell states, and gene expression heterogeneity within tissues

ROC & AUC

	Positive	Negative
Positive	**True Positive Rate**	**False Positive Rate**
Negative	**False Negative Rate**	**True Negative Rate**





- **ROC(Receiver Operating Characteristic)**

- A metric for evaluating the performance of a binary classification model, represented as a graph that visually shows the model's performance across different classification thresholds

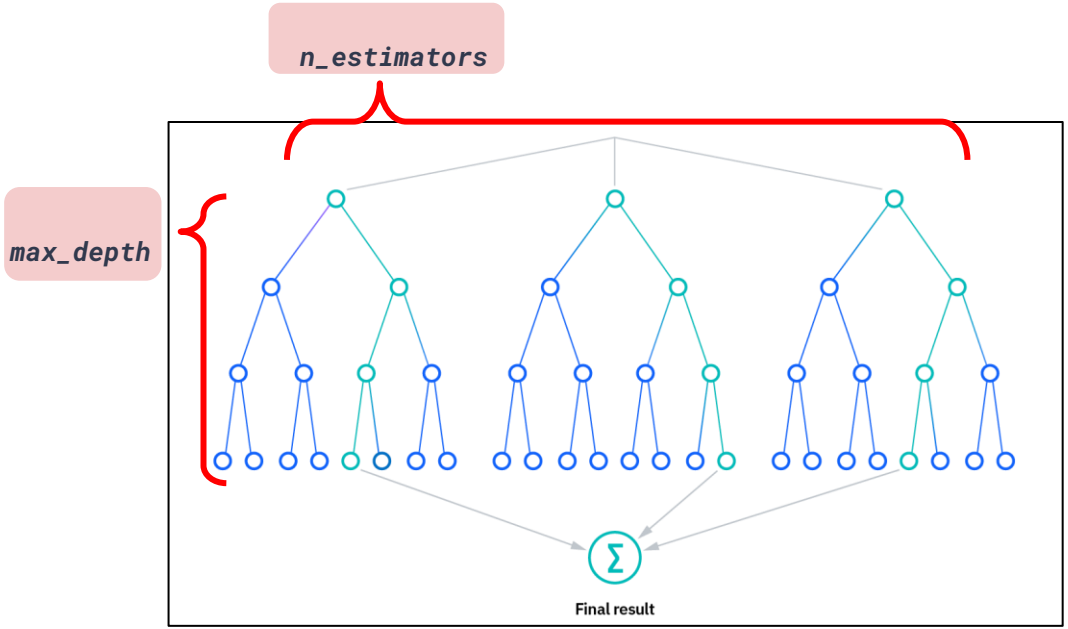
- **X-axis** : the False Positive Rate (FPR)
- **Y-axis** :the True Positive Rate (TPR)
- The closer the curve is to the top-left corner, the better the performance of the model

- **Accuracy** = (TP + TN) / (TP + TN + FP + FN)

- **AUC(Area Under the ROC Curve)**
  Closer to 1 → curve near top-left → better model performance

*Korean J Anesthesiol 2022;75(1):25-36

Random Forest



- **Random Forest**

- Builds multiple decision trees and averages their predictions to produce the final model

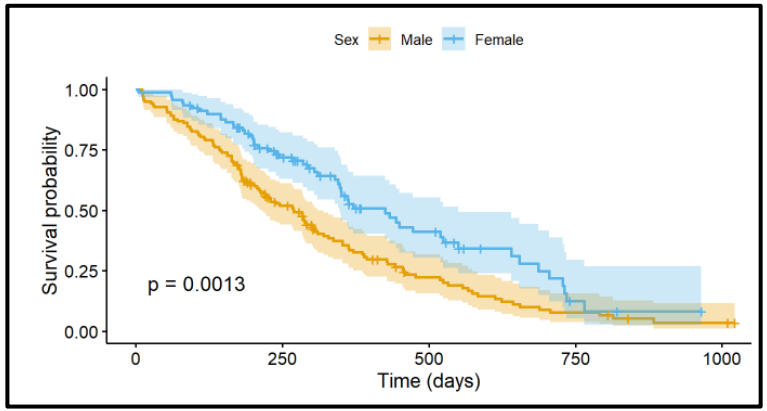
- Each tree uses random subsets of data and features for splitting

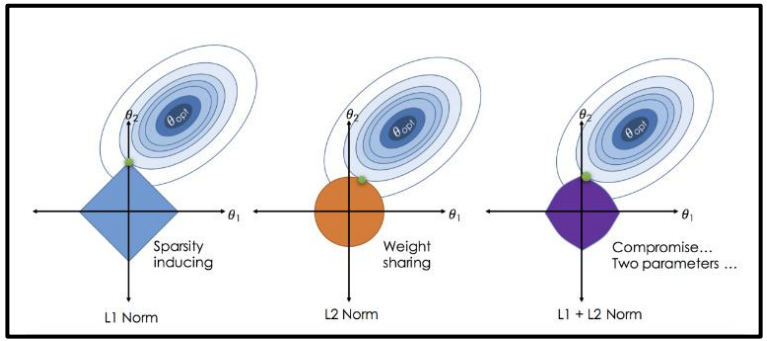
- **Key Parameters**
  a. ***n_estimators***
  b. ***max_depth***
  c. ***max_features, min_samples_split, min_samples_leaf***

*James, G., Witten, D., Hastie, T., Tibshirani, R. (2023). *An Introduction to Statistical Learning* (2nd ed.), Springer, Chapter 8.

*Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

CoxBoost + Enet





- **Cox Proportional Hazards Model (Cox Model)**

- A statistical model used in survival analysis to predict survival based on time-to-event data

- Analyzes the effect of multiple variables on the risk of a specific event (e.g., death, disease occurrence)

- **CoxBoost**
- A model that improves the predictive accuracy of the traditional Cox Model using boosting techniques.

- **Enet (Elastic Net)**
- A regression model that combines L1 (Lasso) and L2 (Ridge) regularization to perform variable selection and regularization simultaneously

scikit-survival 문서: "Penalized Cox Models"

Verification

Table 1.

Description of R packages that implement machine learning algorithms

Algorithms	R packages
(Random survival forest) RSF	RandomForestSRC
(Generalized boosted regression modeling) GBM	gbm
(Least absolute shrinkage and selection operator) Lasso	glmnet
Ridge	glmnet
(Elastic network) Enet	glmnet
CoxBoost	CoxBoost
Stepcox	survival
(Supervised principal components) SuperPC	superpc
(Survival support vector machine) survival-SVM	survivalsvm
(Partial least squares regression for Cox) plsRcox	plsRcox
Boruta	Boruta

- **Identified 83 DC marker genes using an algorithm after integrating single-cell RNA-sequencing and bulk RNA-seq data**

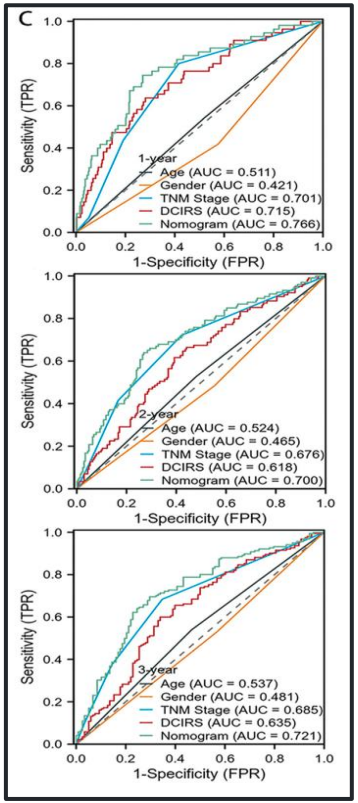
- Built predictive models using 10 algorithms and 90 different combinations

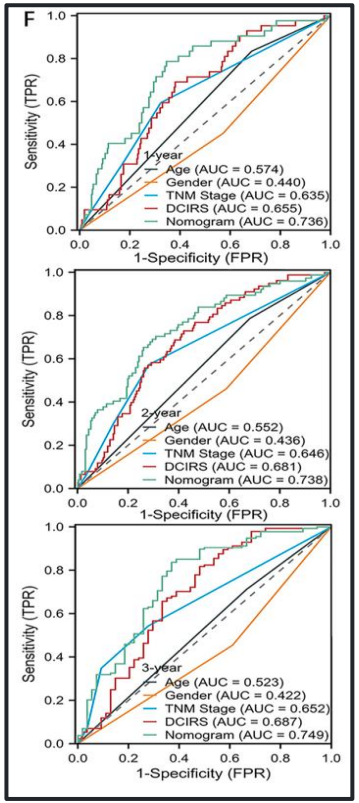
- Among these, the combination of CoxBoost + Enet ($\alpha = 0.2$) with the highest C-index was used in the study

- **DCIRS(Dendritic Cell Implicated Risk Score)**

- Selected 7 key genes from the identified DC marker genes using machine learning to construct DCIRS

Modeling Results_ROC curve & AUC





- Fig. 3C, F: ROC curves evaluating the predictive performance of nomograms combining clinical variables and DCIRS
  (C – TCGA dataset / F – GSE72094 dataset)

- The study validated DCIRS through such analyses, demonstrating its prognostic ability to predict patient survival and the robustness of the model.

Question.

Does training on an optimized combination of features improve machine learning performance compared to learning from diverse information?

Question

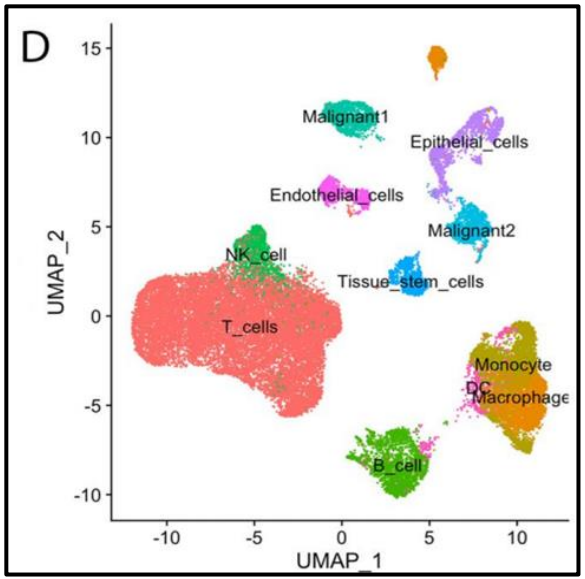
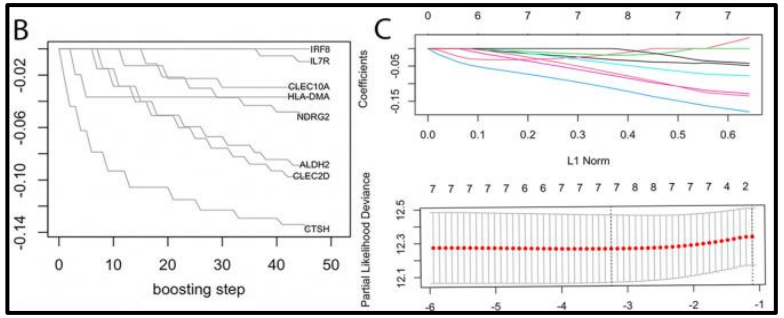


Table S2. DC cell marker genes identified from EMTAB6149.

No.	Gene	No.	Gene	No.	Gene
1	PLD4	32	AIF1	63	PPA1
2	S100B	33	IGSF6	64	SLA
3	CD1C	34	C1orf54	65	IL7R
4	PPP1R14A	35	KCNMB1	66	GIMAP4
5	PKIB	36	IL18	67	EVL
6	CSF2RA	37	CLEC4A	68	PPP2R5C
7	FCER1A	38	GAPT	69	NKG7
8	CD1A	39	HLA-DRB5	70	CLEC2D
9	SERPINF1	40	C12orf45	71	GIMAP7
10	RNASE6	41	FGL2	72	CD69
11	IDO1	42	CST3	73	CTSW
12	CPVL	43	CXorf21	74	PIK3IP1
13	CD1E	44	CLEC10A	75	CD2
14	HLA-DQB2	45	CD86	76	ITM2A
15	PLAC8	46	LY86	77	LCK
16	LGALS2	47	CD83	78	CD3E
17	IRF8	48	TYROBP	79	CD7
18	HCK	49	HLA-DMA	80	TRBC2
19	PAK1	50	DAPP1	81	IL32
20	LST1	51	ALDH2	82	GZMA
21	GSN	52	RAB32	83	CD3D
22	SPI1	53	IFI30		
23	MS4A6A	54	CLIC2		
24	NDRG2	55	CFP		
25	HLA-DQA1	56	MNDA		
26	CALHM6	57	LGALS9		
27	HLA-DQA2	58	CTSH		
28	HLA-DMB	59	CLN8		
29	SMCO4	60	TSPAN13		
30	PLEK	61	GRN		
31	HLA-DQB1	62	CD300A		



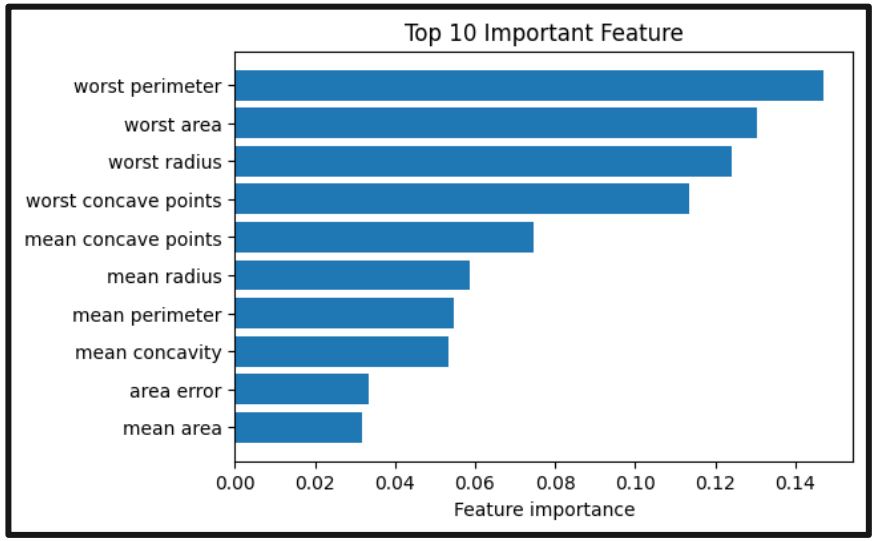
$$\begin{aligned}\text{DCIRS} = {} & \text{IL7R} \times (-0.02084432) + \text{IRF8} \times (-0.01001391) \\ & + \text{CLEC10A} \times (-0.03966908) + \text{CLEC2D} \times (-0.10397921) \\ & + \text{CTSH} \times (-0.1457732) + \text{ALDH2} \times (-0.10492581) \\ & + \text{NDRG2} \times (-0.06235225).\end{aligned}$$

scRNAseq-based 83 marker genes → **7 gene selection**

Why was it necessary to select only 7 genes instead of considering all 83 genes?

Experiment Design

Why was it necessary to select only 7 genes instead of considering all 83 genes?
→ Select number/type of features and run Random Forest



Select feature via Importance

Top 3	Bottom 3
Top 5	Bottom 5
Top 7	Bottom 7
Top 10	Bottom 10
Top 20	Bottom 20

Base
n estimator: 1
max depth: 1

Tuned
(grid search)
n estimator: 100
max depth: 50

Feature 30 learned model from Mission 1
→ Positive Control

Validation with ROC & AUC

Hyper Parameter Tuning

```python
# ------------------------------------------------------------
# 6-e. 하이퍼파라미터 영역 탐색 (n_estimators × max_depth) + 히트맵
# ------------------------------------------------------------
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier

# 1) 데이터 ------------------------------------------------------------
X, y = load_breast_cancer(return_X_y=True)

# 2) 교차검증 분할기 ------------------------------------------------------------
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=77)

# 3) 탐색할 값 리스트 ------------------------------------------------------------
n_list = [1, 50, 100, 200]      # 열(Columns) : 트리 개수
d_list = [1, 50, 100, 200]        # 행(Rows)     : 최대 깊이

# 4) 결과 저장용 행렬 (행 = depth, 열 = tree 수) ---------------------------
heat = np.empty((len(d_list), len(n_list)))

for row, depth in enumerate(d_list):
    for col, trees in enumerate(n_list):
        model = RandomForestClassifier(
            n_estimators     = trees,
            max_depth        = depth,
            random_state     = 77,
        )
        # Stratified 5-fold ROC-AUC 평균
        score = cross_val_score(
            model, X, y,
            cv      = cv,
            scoring = "roc_auc",   # 문자열 지정이 가장 안전
            n_jobs  = -1,
        ).mean()
        heat[row, col] = score

# 5) 히트맵 ------------------------------------------------------------
fig, ax = plt.subplots(figsize=(8, 6))
im = ax.imshow(heat, cmap="viridis", origin="upper")
```

- **Data Preparation**
  `X, y = load_breast_cancer(return_X_y=True)`
  → load scikit-learn breast cancer dataset

- **Cross-Validation Setup**
  `cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=77)`
  → 5-fold CV maintaining class ratio

- **Hyperparameter Lists**
  `n_list = [1, 50, 100, 200]` → number of trees (columns)
  `d_list = [1, 50, 100, 200]` → max depth (rows)

- **Result Array**
  `heat = np.empty((len(d_list), len(n_list)))`
  → store AUC scores for each combination

- **Nested For Loop**
  `for row, depth in enumerate(d_list):`
  `for col, trees in enumerate(n_list):`
  → Iterate over all combinations

- **Define the Random Forest model**
  `model = RandomForestClassifier(n_estimators=trees, max_depth=depth, ...)`

- **Perform cross-validation (ROC-AUC)**
  `score = cross_val_score(..., scoring="roc_auc", cv=cv).mean()`
  → Compute 5-fold mean AUC for each model combination

- Store the results
  `heat[row, col] = score` → Record the score at the corresponding position

```
38
39  # 5) 히트맵 ------------------------------------------------------------------
40  fig, ax = plt.subplots(figsize=(8, 6))
41  im = ax.imshow(heat, cmap="viridis", origin="upper")

    - -
 1  # 축 레이블·눈금
 2  ax.set_xticks(range(len(n_list)))
 3  ax.set_yticks(range(len(d_list)))
 4  ax.set_xticklabels(n_list)                # n_estimators 값
 5  ax.set_yticklabels([str(d) for d in d_list])   # max_depth 값
 6  ax.set_xlabel("n_estimators")
 7  ax.set_ylabel("max_depth")
 8  ax.set_title("ROC-AUC Heatmap (Stratified 5-fold, random_state = 77)")
 9
10  # 셀마다 점수 표시
11  for row in range(len(d_list)):
12      for col in range(len(n_list)):
13          ax.text(
14              col, row, f"{heat[row, col]:.3f}",
15              ha="center", va="center",
16              color="white" if heat[row, col] < 0.90 else "black",
17              fontsize=8,
18          )
19
20  # 컬러바 + 레이아웃
21  fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
22  plt.tight_layout()
23  plt.show()
24  # ---------------------------- (End of 6-e 블록) ----------------------------
```

fig, ax : figure (entire plot) and axis objects
figsize=(8,6) : plot size in inches (width × height)
cmap="viridis"color map;
higher values shown in brighter colors
origin="upper": draw 2D array starting from the top

Set Axis Ticks & Labels
- Use list indices for x/y tick positions
- Display actual hyperparameter values (n_estimators, max_depth) as tick labels

Annotate Cells
- Display ROC-AUC score in each heatmap cell (3 decimal places)
- Use white text for low scores, black text for high scores
  → improves readability

Layout & Display
- tight_layout() : automatically adjust spacing to prevent overlapping plot elements
- show(): display the final heatmap on screen

Hyper Parameter Tunning

```python
# ------------------------------------------------------------
# 6-g. ROC curve (Baseline & Tuned) + AUC Heatmap
# ------------------------------------------------------------
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# ---------- 1) Feature Set 정의 ----------------------------------------
sizes        = [3, 5, 7, 10, 20]
order_labels = ["All30"] \
             + [f"Top{n}"    for n in sizes] \
             + [f"Bottom{n}" for n in sizes]

feature_sets = {
    "All30":     np.arange(X.shape[1]),
    **{f"Top{n}"    : indices_desc[:n] for n in sizes},
    **{f"Bottom{n}": indices_asc[:n]  for n in sizes},
}
```

1) Define Feature Set

sizes : list of feature counts to use in experiments

order_labels : labels for sorting results

"All30" → use all 30 features

"TopN" → use top N features by importance

"BottomN" → use bottom N features by importance

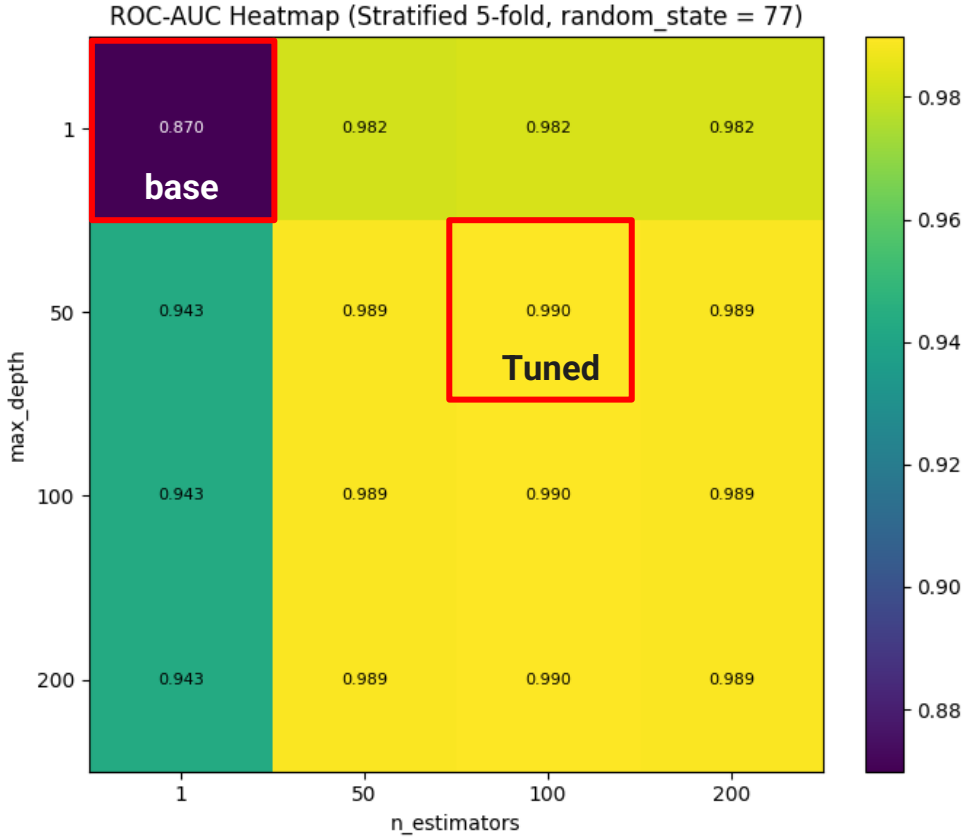
Hyper Parameter Tunning

```
20  # ---------- 2) 두 모델 ----------
21  baseline_clf = RandomForestClassifier(
22      n_estimators = 1,
23      max_depth    = 1,
24      random_state = 77,
25  )
26  tuned_clf = RandomForestClassifier(
27      n_estimators = 100,
28      max_depth    = 50,
29      random_state = 77,
30  )
```

2) Define Two Models

baseline_clf : very simple Random Forest (1 tree, depth 1)
→ checks baseline performance
weak model with minimal complexity

tuned_clf: optimized model (100 trees, depth 50)
트리 수 100개, 깊이 50
→ significantly better performance than baseline
used for actual ROC curve comparison



Hyper Parameter Tunning

```python
# ---------- 3) ROC 데이터 & AUC 저장 ----------
roc_base, roc_tuned = {}, {}
auc_mat = np.empty((len(order_labels), 2))  # 열 0=Base, 1=Tuned

for i, label in enumerate(order_labels):
    idx = feature_sets[label]

    # — 학습
    baseline_clf.fit(X_train[:, idx], y_train)
    tuned_clf.fit(X_train[:, idx], y_train)

    # — 예측 확률
    pb = baseline_clf.predict_proba(X_test[:, idx])[:, 1]
    pt = tuned_clf.predict_proba(X_test[:, idx])[:, 1]

    # — ROC
    fpr_b, tpr_b, _ = roc_curve(y_test, pb)
    fpr_t, tpr_t, _ = roc_curve(y_test, pt)
    auc_b, auc_t    = auc(fpr_b, tpr_b), auc(fpr_t, tpr_t)

    roc_base[label]  = (fpr_b, tpr_b, auc_b)
    roc_tuned[label] = (fpr_t, tpr_t, auc_t)
    auc_mat[i]       = [auc_b, auc_t]

```

3) Compute ROC Data & Store AUC

order_labels : : names of feature combinations
("All30", "Top3", "Bottom3")

idx : feature indices to use for the corresponding combination

```python
# ---------- 4) ROC Curve A : All30 + Top N ------------------------------
top_labels = ["All30"] + [f"Top{n}" for n in sizes]
colors = plt.cm.tab10(np.linspace(0, 1, len(top_labels)))

plt.figure(figsize=(7, 7))
for c, lbl in zip(colors, top_labels):
    fpr_b, tpr_b, A_b = roc_base[lbl]
    fpr_t, tpr_t, A_t = roc_tuned[lbl]
    plt.plot(fpr_b, tpr_b, linestyle="--", color=c,
             label=f"{lbl} Base ({A_b:.3f})")
    plt.plot(fpr_t, tpr_t, linestyle="-",  color=c,
             label=f"{lbl} Tuned ({A_t:.3f})")
plt.plot([0, 1], [0, 1], "k:", lw=1)
plt.title("ROC – All30 + Top N (Baseline vs Tuned)")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend(fontsize=7, ncol=2, loc="lower right")
plt.grid(alpha=0.3); plt.tight_layout(); plt.show()

```

4) ROC Curve Visualization – Top N Combinations

top_labels = ["All30"] + [f"Top{n}" for n in sizes]
select full + top feature combinations for visualization

colors = plt.cm.tab10(np.linspace(0, 1, len(top_labels)))
Distribute colors from Matplotlib's 10 default colors according to the number of matplotlib

plt.plot([0, 1], [0, 1], "k:", lw=1)
Baseline (ROC of random prediction)

```python
# ---------- 5) ROC Curve B : All30 + Bottom N ---------------------------
bottom_labels = ["All30"] + [f"Bottom{n}" for n in sizes]
colors = plt.cm.tab20(np.linspace(0, 1, len(bottom_labels)))

plt.figure(figsize=(7, 7))
for c, lbl in zip(colors, bottom_labels):
    fpr_b, tpr_b, A_b = roc_base[lbl]
    fpr_t, tpr_t, A_t = roc_tuned[lbl]
    plt.plot(fpr_b, tpr_b, linestyle="--", color=c,
             label=f"{lbl} Base ({A_b:.3f})")
    plt.plot(fpr_t, tpr_t, linestyle="-",  color=c,
             label=f"{lbl} Tuned ({A_t:.3f})")
plt.plot([0, 1], [0, 1], "k:", lw=1)
plt.title("ROC – All30 + Bottom N (Baseline vs Tuned)")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend(fontsize=7, ncol=2, loc="lower right")
plt.grid(alpha=0.3); plt.tight_layout(); plt.show()
```

5) ROC Curve B – All30 + Bottom N

- Extract ROC curve data (roc_base, roc_tuned) for each feature combination
- Visualize performance of Baseline (simple) vs Tuned (optimized) models
- Each curve shows AUC scores
- Compare results of All30 vs Bottom N feature combinations

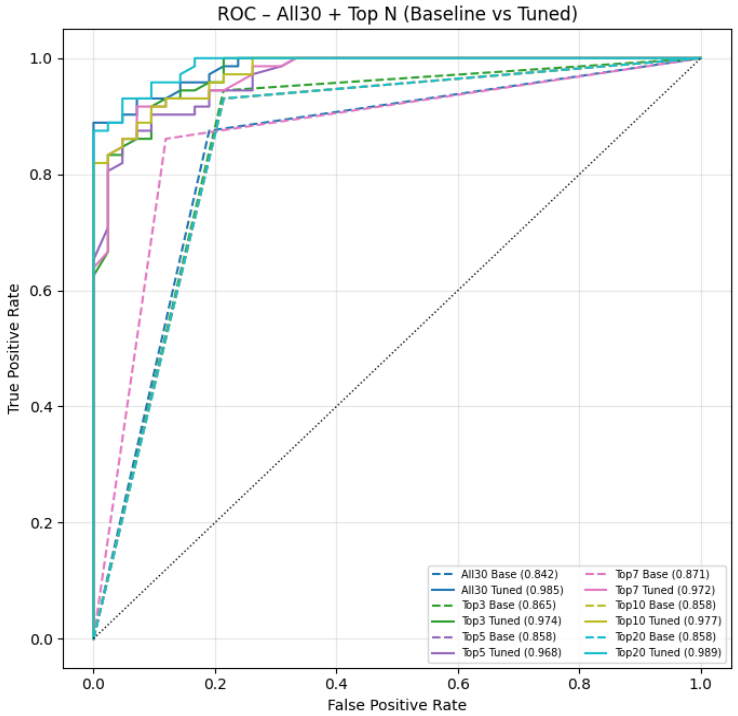
```python
# ---------- 5) ROC Curve B : All30 + Bottom N ----------
bottom_labels = ["All30"] + [f"Bottom{n}" for n in sizes]
colors = plt.cm.tab20(np.linspace(0, 1, len(bottom_labels)))

plt.figure(figsize=(7, 7))
for c, lbl in zip(colors, bottom_labels):
    fpr_b, tpr_b, A_b = roc_base[lbl]
    fpr_t, tpr_t, A_t = roc_tuned[lbl]
    plt.plot(fpr_b, tpr_b, linestyle="--", color=c,
             label=f"{lbl} Base ({A_b:.3f})")
    plt.plot(fpr_t, tpr_t, linestyle="-",  color=c,
             label=f"{lbl} Tuned ({A_t:.3f})")
plt.plot([0, 1], [0, 1], "k:", lw=1)
plt.title("ROC – All30 + Bottom N (Baseline vs Tuned)")
plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
plt.legend(fontsize=7, ncol=2, loc="lower right")
plt.grid(alpha=0.3); plt.tight_layout(); plt.show()

# ---------- 6) AUC Heatmap ----------
fig, ax = plt.subplots(figsize=(6, 8))
im = ax.imshow(auc_mat, cmap="coolwarm", aspect="auto",
               vmin=auc_mat.min(), vmax=auc_mat.max())

ax.set_xticks([0, 1]); ax.set_xticklabels(["Baseline", "Tuned"])
ax.set_yticks(np.arange(len(order_labels))); ax.set_yticklabels(order_labels)
ax.set_title("AUC Heatmap")

best = auc_mat.max()
for r in range(len(order_labels)):
    for c in range(2):
        v = auc_mat[r, c]
        ax.text(c, r, f"{v:.3f}",
                ha="center", va="center",
                color="white" if v < (auc_mat.min()+auc_mat.max())/2 else "black",
                fontsize=7)

fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
plt.tight_layout(); plt.show()
# ---------- (End of 6-g 블록) ----------
```

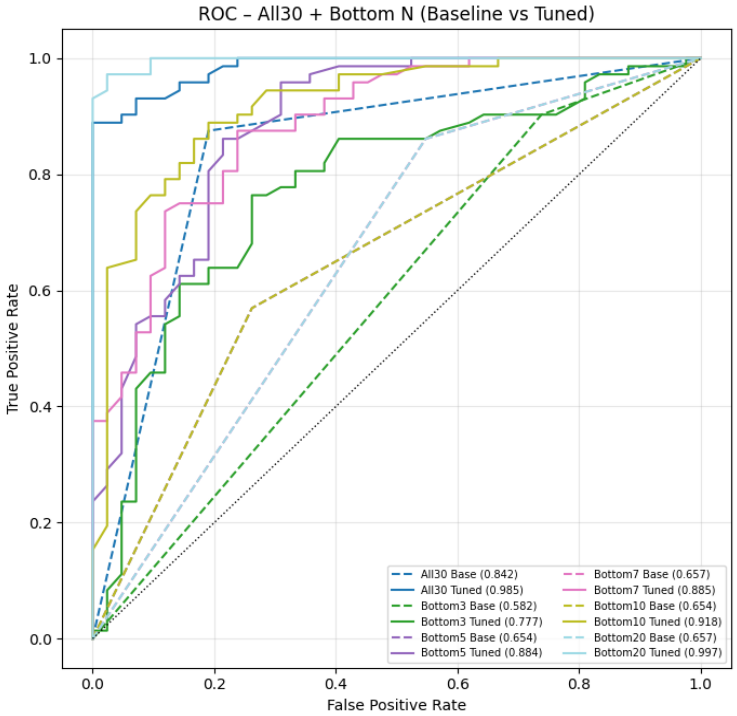
5) ROC Curve B – All30 + Bottom N

- Extract ROC curve data (roc_base, roc_tuned) for each feature combination
- Visualize performance of Baseline (simple) vs Tuned (optimized) models
- Each curve shows AUC scores
- Compare results of All30 vs Bottom N feature combinations

6) AUC Heatmap

- Store AUC scores of all feature combinations (All30, TopN, BottomN) in a 2D matrix
- Compare Baseline and Tuned models via color intensity
- Display AUC values directly in each cell → intuitive summary of model performance





[All 30] 0.842 → 0.985 : Improvement achieved through hyperparameter tuning ▲ (Alleviation of underfitting)

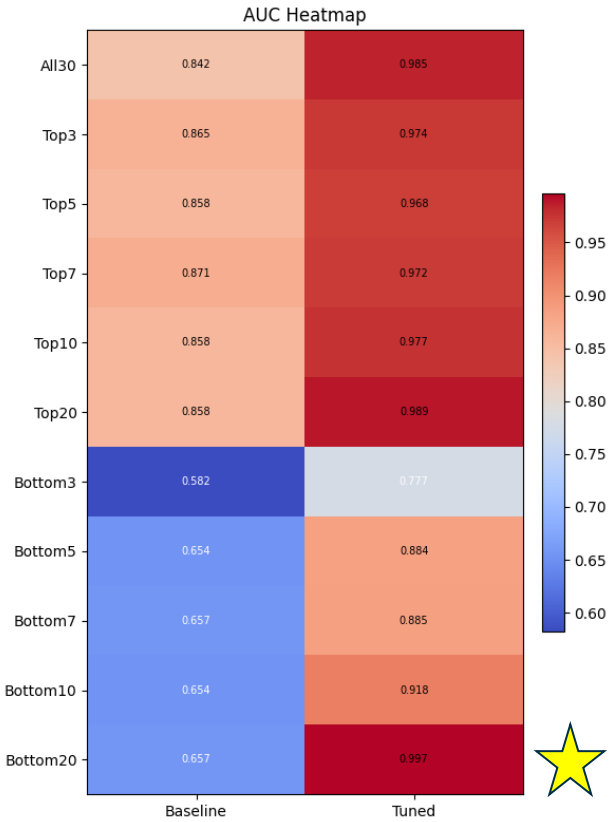
[Top] base & tune show comparable performance using all 30 features / top 10 & 20: slight improvement over base with all 30

➜ Predictive power maintained using only top important features

[Bottom] Base performance is low (insufficient information) / bottom 20: Improvement through hyperparameter tuning ▲

➜ **Even weak variables can recover performance through increasing their number and tuning.**

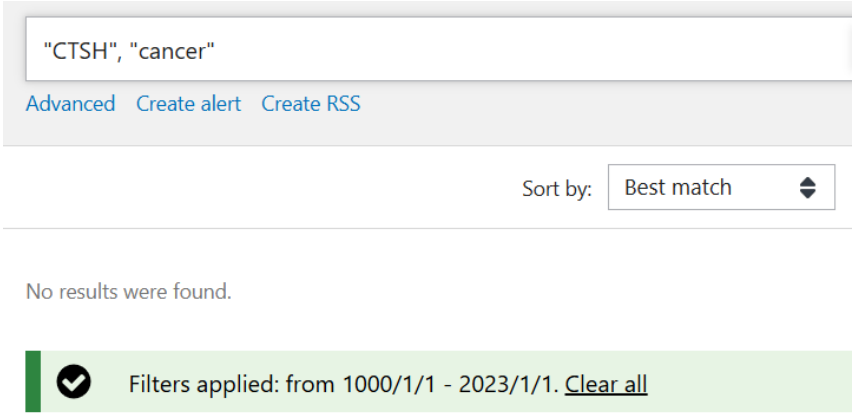
Serendipity

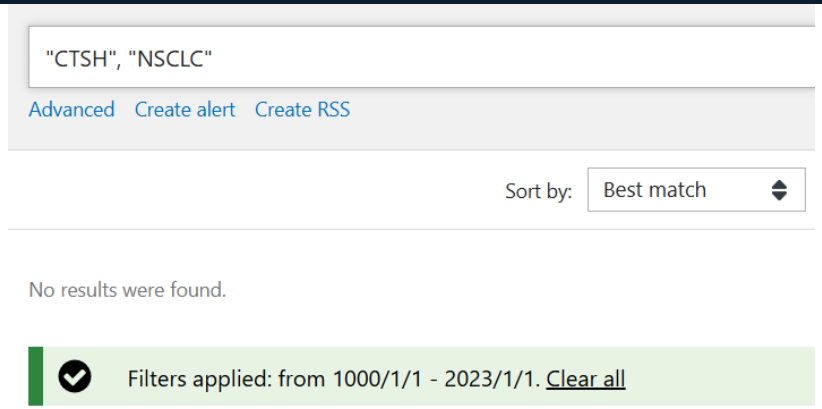


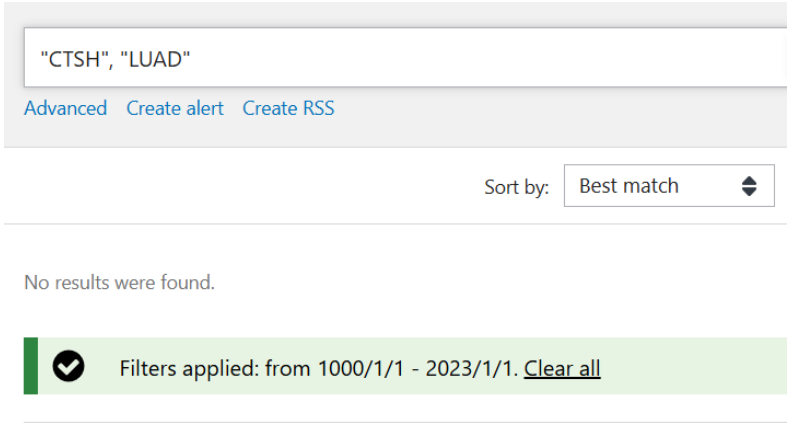
- **Bottom 20 (tuned) :** AUC 0.657 → 0.997
  ➜ Achieved the highest performance among all models
- Even **low-importance variables** can exhibit hidden synergy through hyperparameter optimization
- Performance reversal that could not be predicted by feature importance alone

Serendipity - CTSH

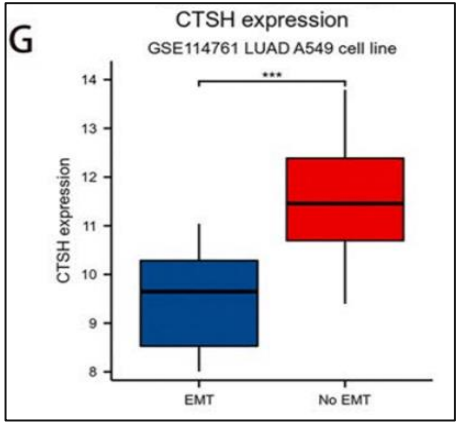
$$\begin{aligned}\text{DCIRS} = {} & \text{IL7R} \times (-0.02084432) + \text{IRF8} \times (-0.01001391) \\ & + \text{CLEC10A} \times (-0.03966908) + \text{CLEC2D} \times (-0.10397921) \\ & + \text{CTSH} \times (-0.1457732) + \text{ALDH2} \times (-0.10492581) \\ & + \text{NDRG2} \times (-0.06235225).\end{aligned}$$

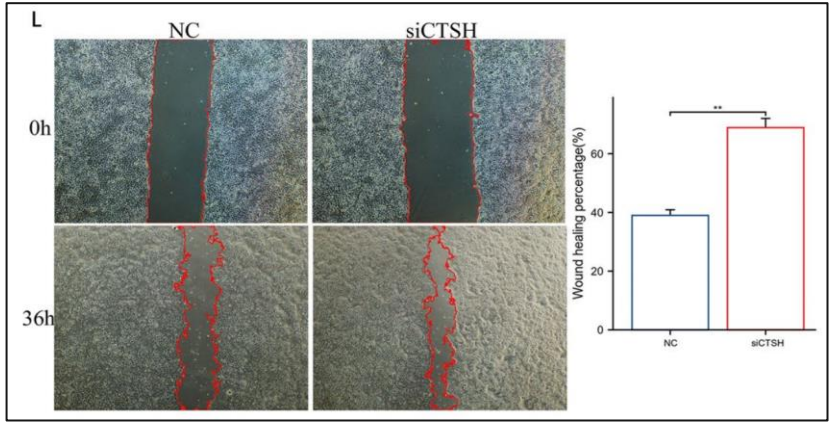


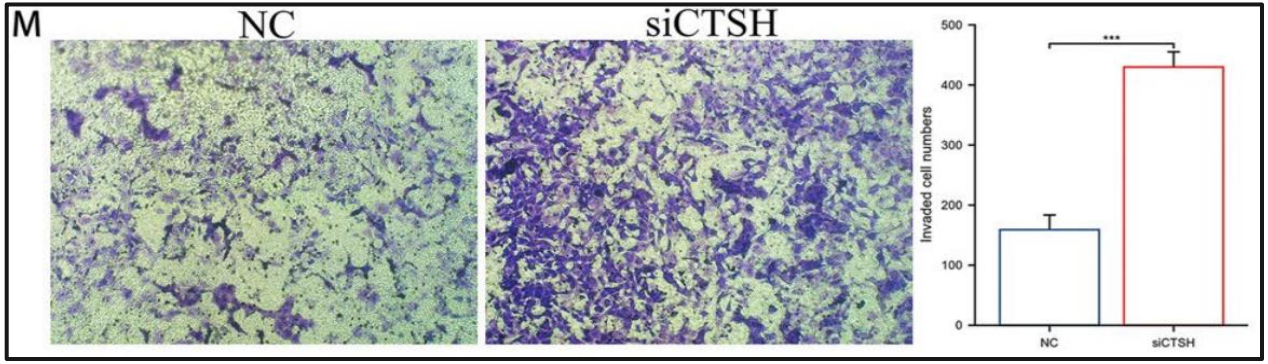




Serendipity - CTSH







- The role of CTSH in cancer remains unclear
- Machine learning analysis based on LUAD transcriptome
  → suggests associations with risk factors
- In vitro validation →
  functions in EMT,
  wound healing,
  and immune evasion mechanisms

Discussion

- **Limited clinical validation**
  Models developed and validated using only public cohorts (TCGA, GEO)
  → predictive performance on real patient data not guaranteed

- **Data type consideration**
  Analysis appears closer to microarray rather than bulk RNA-seq

- **Reproducibility issue**
  CoxBoost used for gene selection is a discontinued package
  → results hard to reproduce

Python Coding Mission II

Python Coding Mission II

1. Count the number of ‘setosa’, ‘versicolor’, and ‘virginica’ species in iris.csv and save the results to Results.txt.

```
with open("iris.csv", "r") as file:
    header = file.readline().strip().split(",")
    for line in file:
        values = line.strip().split(",")
        all_data.append(dict(zip(header, values)))
```

```
species_count = {}

for row in all_data:
    species = row["species"]
    species_count[species] = species_count.get(species, 0) + 1
```

```
with open("Results.txt", "w") as file:
    file.write(
        ", ".join(f"{species}={count}" for species, count in species_count.items())
    )
    file.write("\n")
```

1) Organize the entire dataset as a list of dictionaries

- ***species_count = {}***
  Use an empty dictionary to store species counts → automatically handles new species
- ***.get(species, 0) :*** Iterate through all_data species values
  if not in dictionary → set 0; if exists → add 1 → update dictionary
- ***.items()*** : Return the (key, value) pairs of the dictionary as tuples

Results.txt

```
1    setosa=50, versicolor=50, virginica=50
```

Python Coding Mission II

1. Count the number of ‘setosa’, ‘versicolor’, and ‘virginica’ species in iris.csv and save the results to Results.txt.

```
setosa_filt = 0
versicolor_filt = 0
virginica_filt = 0

for line in all_data:
    if line["species"] == "setosa":
        setosa_filt += 1
    elif line["species"] == "versicolor":
        verisicolor_filt += 1
    else:
        viriginica_filt += 1

with open("Result.txt", "w") as file:
    file.write(f"setosa = {setosa_filt}, versicolor = {versicolor_filt}, virginica = {virginica_filt}")
```

2) Create variables for the counts of each species

- Count using if/elif statements
- Requires initializing variables
- Simple structure, but code must be modified if species change

```
species_count = {"setosa": 0, "versicolor": 0, "virginica": 0}
for row in all_data:
    species = row["species"]
    if species in species_count:
        species_count[species] += 1
with open("Results.txt", "w") as file:
    file.write( ", ".join(f"{species}={count}" for species, count in species_count.items()))
```

3) Predefine species names

- Species not in the dictionary are ignored → limited flexibility

Python Coding Mission II

2. Count the number of species in iris.csv satisfying sepal_length ≥ 4.8 and petal_length > 2, and append the results to Results.txt

```
filtered_species_count = {species: 0 for species in species_count}
for row in all_data:
    sepal_length = float(row["sepal_length"])
    petal_length = float(row["petal_length"])
    species = row["species"]

    if sepal_length >= 4.8 and petal_length > 2:
        filtered_species_count[species] += 1
```

```
with open("Results.txt", "a") as file:
    file.write(
        ", ".join(
            f"{species}={count}" for species, count in filtered_species_count.items()
        )
    )
    file.write("\n")
```

- ***{species: for species in species_count}***
  : Based on species_count dictionary
  → create a new dictionary for each species with initial value 0
- ***float()***: Strings → numeric type for comparison
- +1 to species count only when all conditions are met
- ***with open ("Results.txt", "a")***
  : Append results to the output file

Results.txt

```
1  setosa=50, versicolor=50, virginica=50
2  setosa=0, versicolor=50, virginica=50
```

Python Coding Mission II

3. Append rows from iris.csv where sepal_width > 3, petal_length < 6, and petal_width ≥ 0.8 to Results.txt

```
filtered_rows = []

for row in all_data:
    sepal_width = float(row["sepal_width"])
    petal_length = float(row["petal_length"])
    petal_width = float(row["petal_width"])

    if sepal_width > 3 and petal_length < 6 and petal_width >= 0.8:
        filtered_rows.append(row)
```

- **filtered_rows = [ ] :**
  Create an empty list to store data that meets the conditions
- ***.float()*** **:** Convert strings → numeric (float)
- **.append(row) :** Add to filtered_rows only if all conditions are met

```
with open("Results.txt", "a") as file:

    file.write("\t".join(header) + "\n")
    for row in filtered_rows:
        file.write("\t".join(row[col] for col in header) + "\n")
```

- ***with open ("Results.txt", "a")*** **:** Add to filtered_rows only if all conditions are met
- Each row in filtered_rows is a dictionary;
  access values using row[col] and join with **\t** using **.join()**

Python Coding Mission II

3. **Append rows from iris.csv where sepal_width > 3, petal_length < 6, and petal_width ≥ 0.8 to Results.txt**

```
filtered_rows = []

for row in all_data:
    sepal_width = float(row["sepal_width"])
    petal_length = float(row["petal_length"])
    petal_width = float(row["petal_width"])

    if sepal_width > 3 and petal_length < 6 and petal_width >= 0.8:
        filtered_rows.append(row)
```

```
with open("Results.txt", "a") as file:

    file.write("\t".join(header) + "\n")
    for row in filtered_rows:
        file.write("\t".join(row[col] for col in header) + "\n")
```

Results.txt

```
setosa=50, versicolor=50, virginica=50
setosa=0, versicolor=50, virginica=50
sepal_length    sepal_width petal_length    petal_width species
7   3.2 4.7 1.4 versicolor
6.4 3.2 4.5 1.5 versicolor
6.9 3.1 4.9 1.5 versicolor
6.3 3.3 4.7 1.6 versicolor
6.7 3.1 4.4 1.4 versicolor
5.9 3.2 4.8 1.8 versicolor
6   3.4 4.5 1.6 versicolor
6.7 3.1 4.7 1.5 versicolor
6.5 3.2 5.1 2   virginica
6.4 3.2 5.3 2.3 virginica
6.9 3.2 5.7 2.3 virginica
6.7 3.3 5.7 2.1 virginica
6.3 3.4 5.6 2.4 virginica
6.4 3.1 5.5 1.8 virginica
6.9 3.1 5.4 2.1 virginica
6.7 3.1 5.6 2.4 virginica
6.9 3.1 5.1 2.3 virginica
6.8 3.2 5.9 2.3 virginica
6.7 3.3 5.7 2.5 virginica
6.2 3.4 5.4 2.3 virginica
```

Supplementary

Supplementary

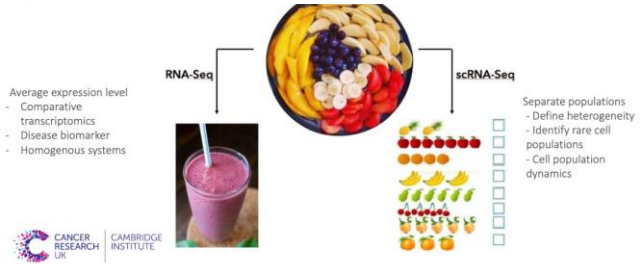
- **Recall, Accuracy**

재현율 (Recall)	정확도 (Accuracy)
True positive rate (sensitivity)	Overall accuracy (correctly classified proportion)
TP / (TP + FN)	(TP + TN) / (TP + FP + TN + FN)

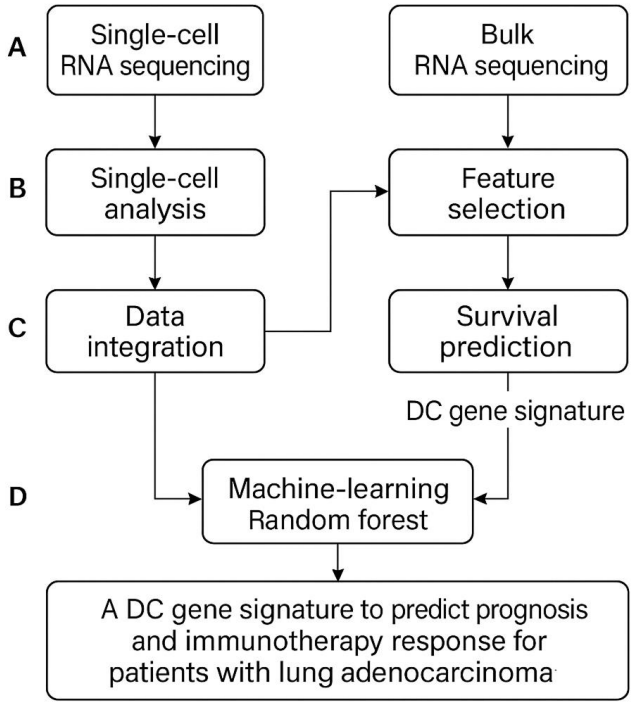
- **F1-score**

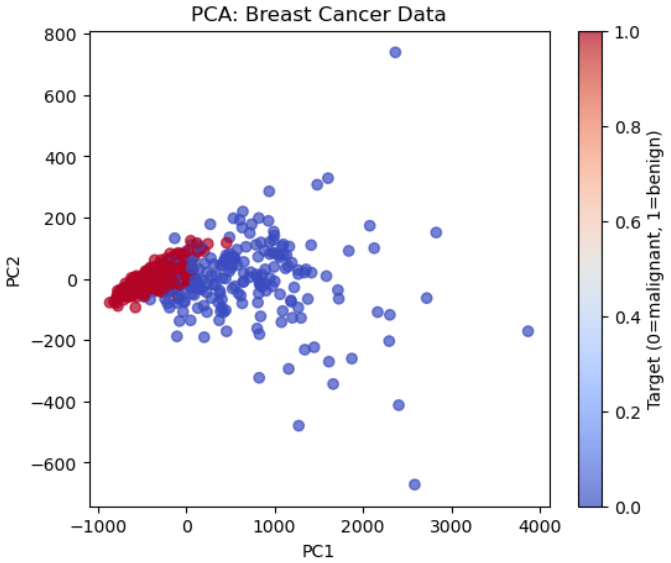
$$\text{F1-score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

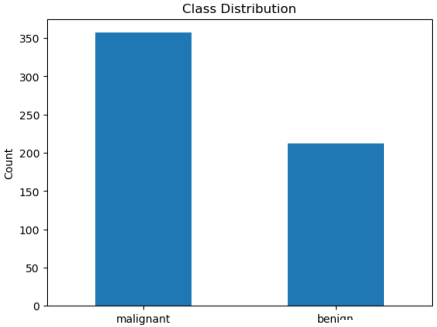
- **Bulk VS Single Cell RNA-Seq**

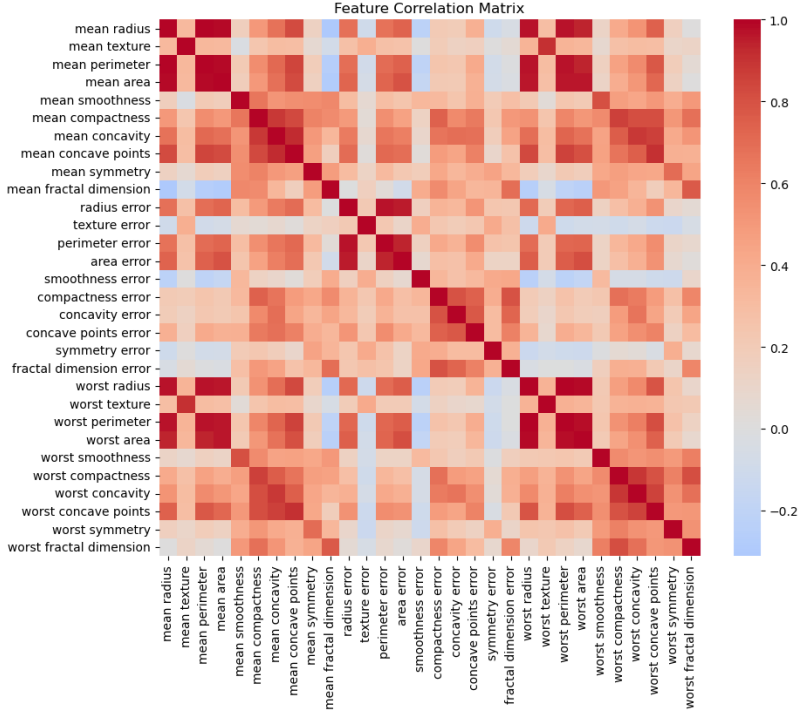


- **Experiment flow**










```python
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=10000)
scores = cross_val_score(model, X, y, cv=5)

print("🔷 로지스틱 회귀 평균 정확도:", scores.mean())
# 만약 로지스틱 회귀로도 90% 이상 정확도가 나온다면,
# 고차원 비선형 모델 없이도 잘 맞는 → 즉, noise가 적고 분류가 잘 되는 구조일 가능성이 높음
```

```
로지스틱 회귀 평균 정확도: 0.9507995652848935
```