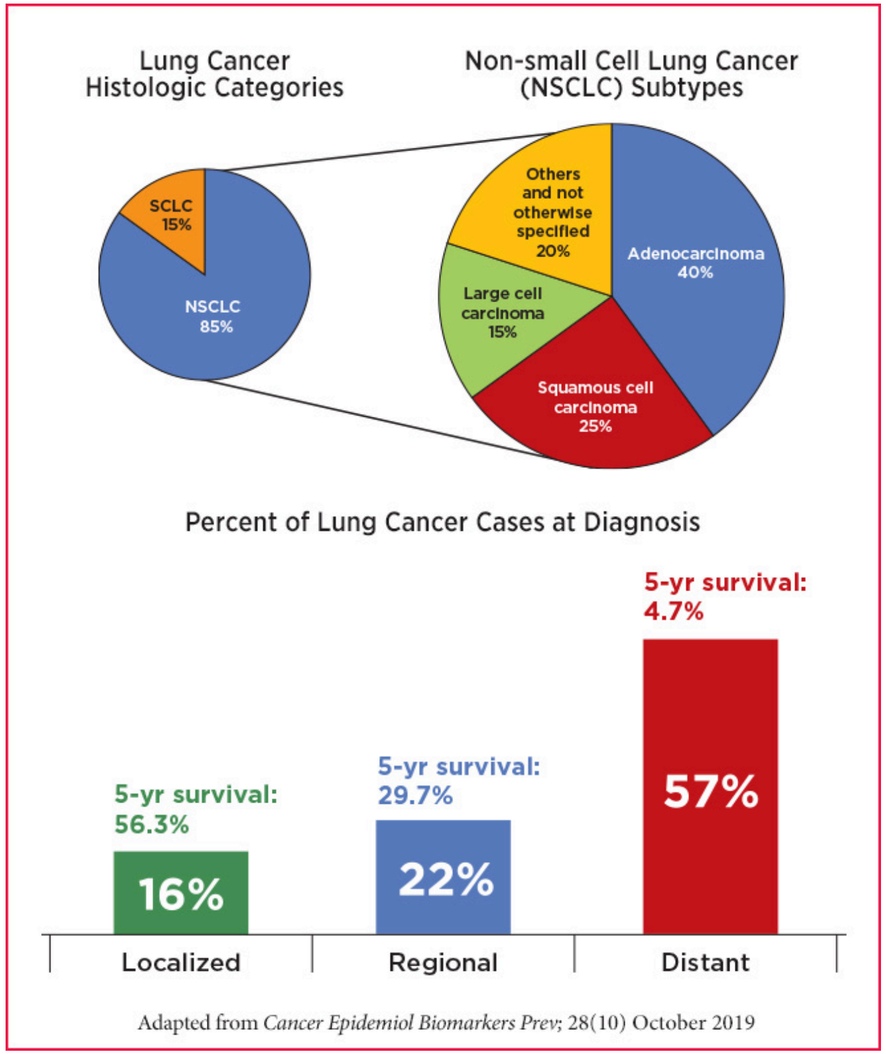
Cancer Biomarkers Discovery and Modelling

Outline

Background

Cancer research importance

- Lung Cancer is the leading death cause in Taiwan
- Low survival rate

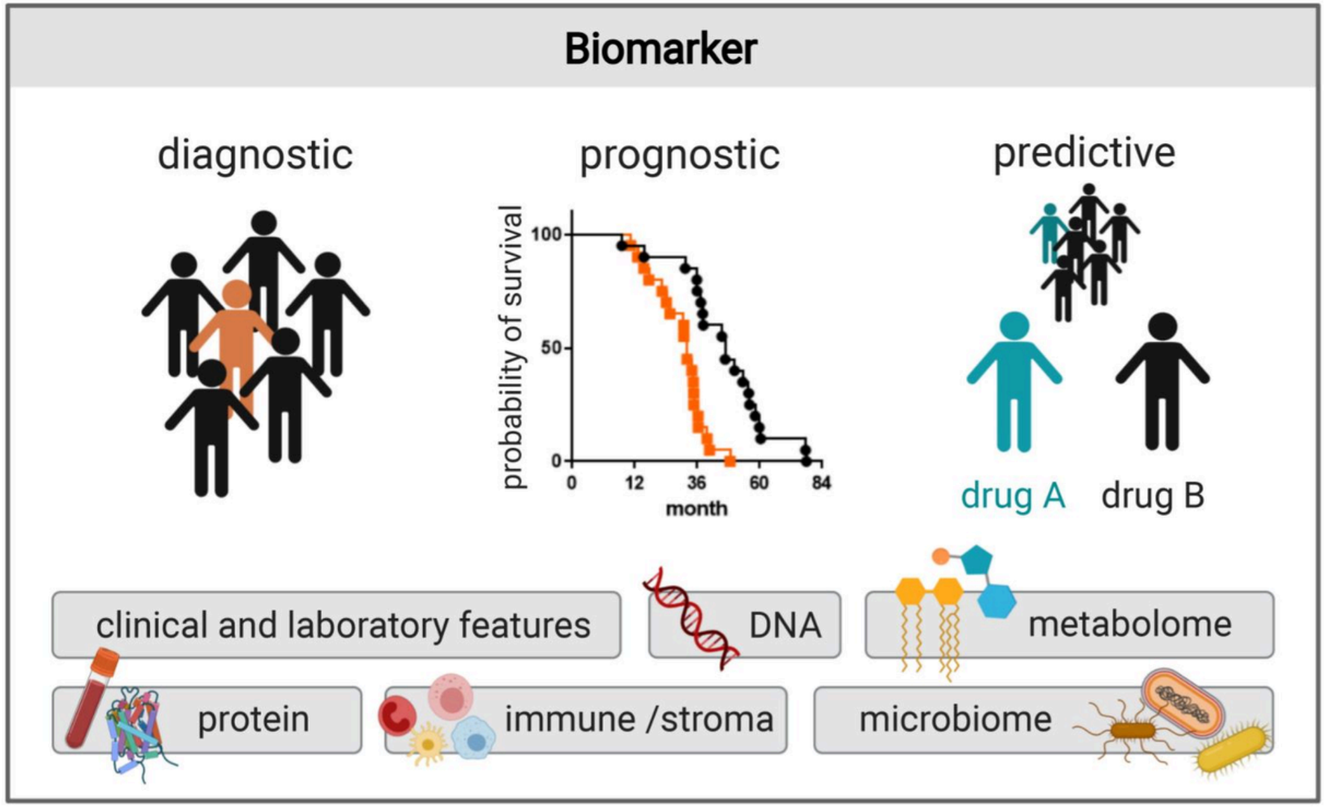


Biomarkers

Biomarker testing is a way to look for genes, proteins, and other substances (called biomarkers or tumor markers) that can provide information about cancer.

Biomarkers can be used to:

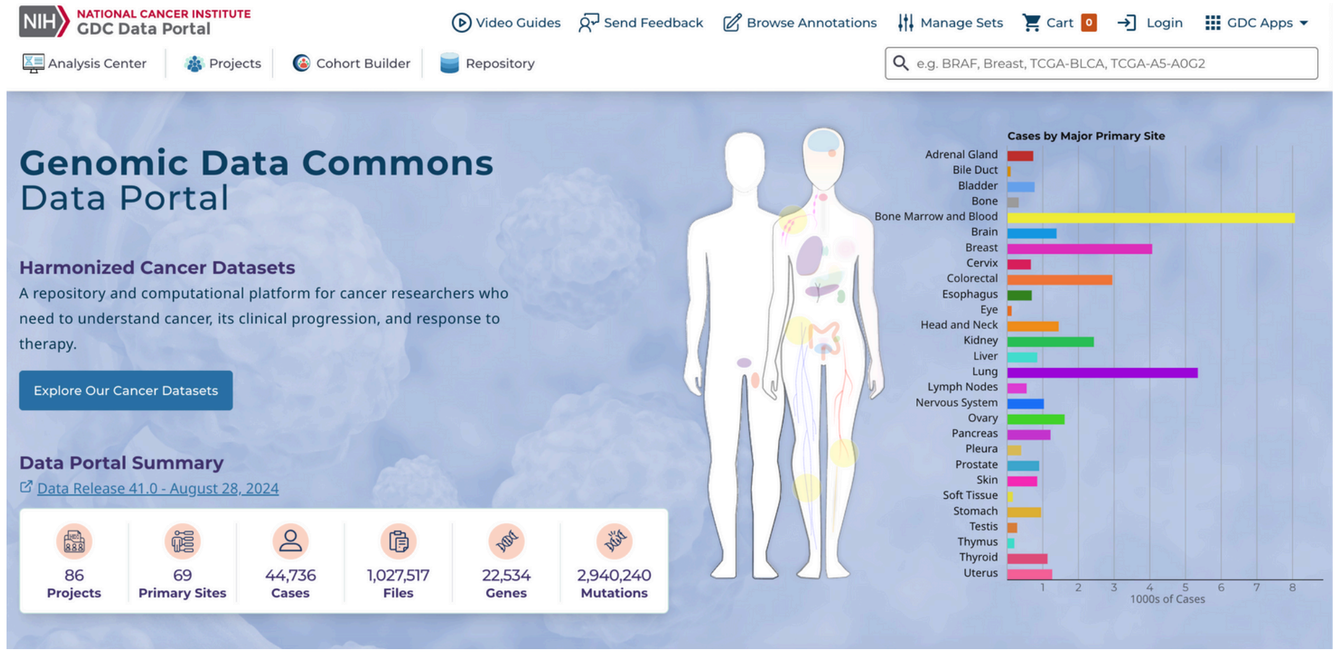
- Diagnose cancer
- Monitor the progression of the disease
- Predict how well a patient will respond to treatment
- Identify new treatments for cancer



Database

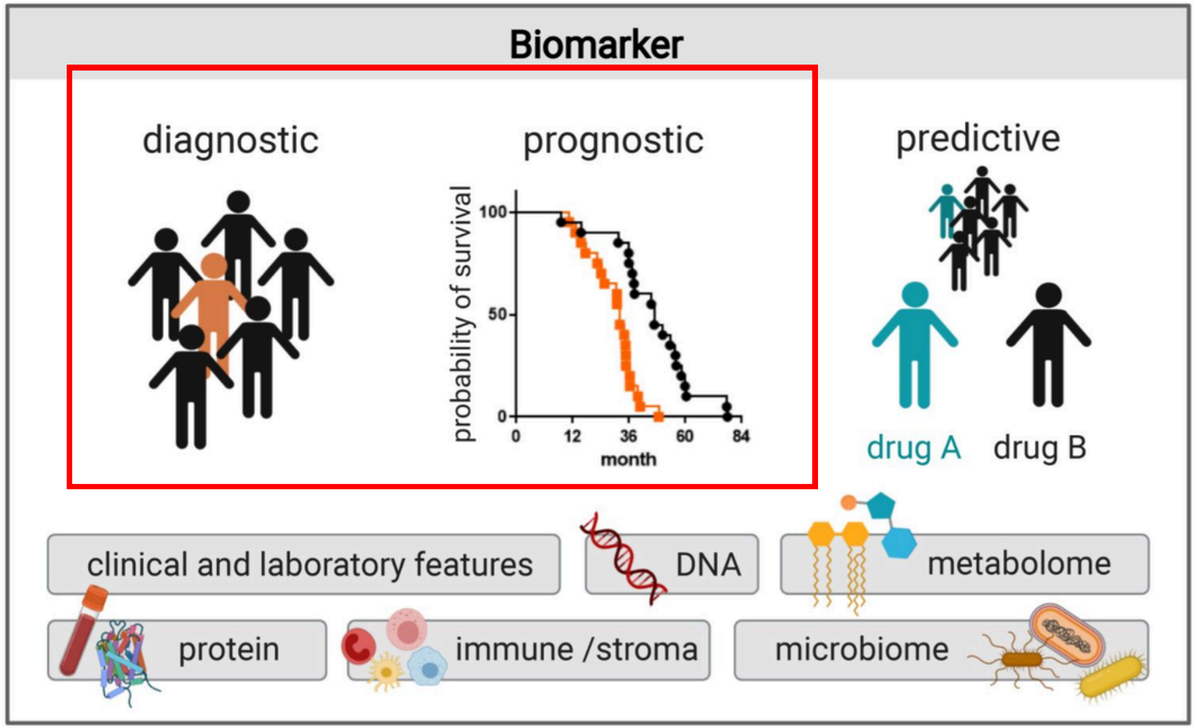
TCGA database:

- Cancer genomics program, investigated primary cancer samples and matched normal samples, including 33 cancer types
- Includes RNA gene expression data of tumor

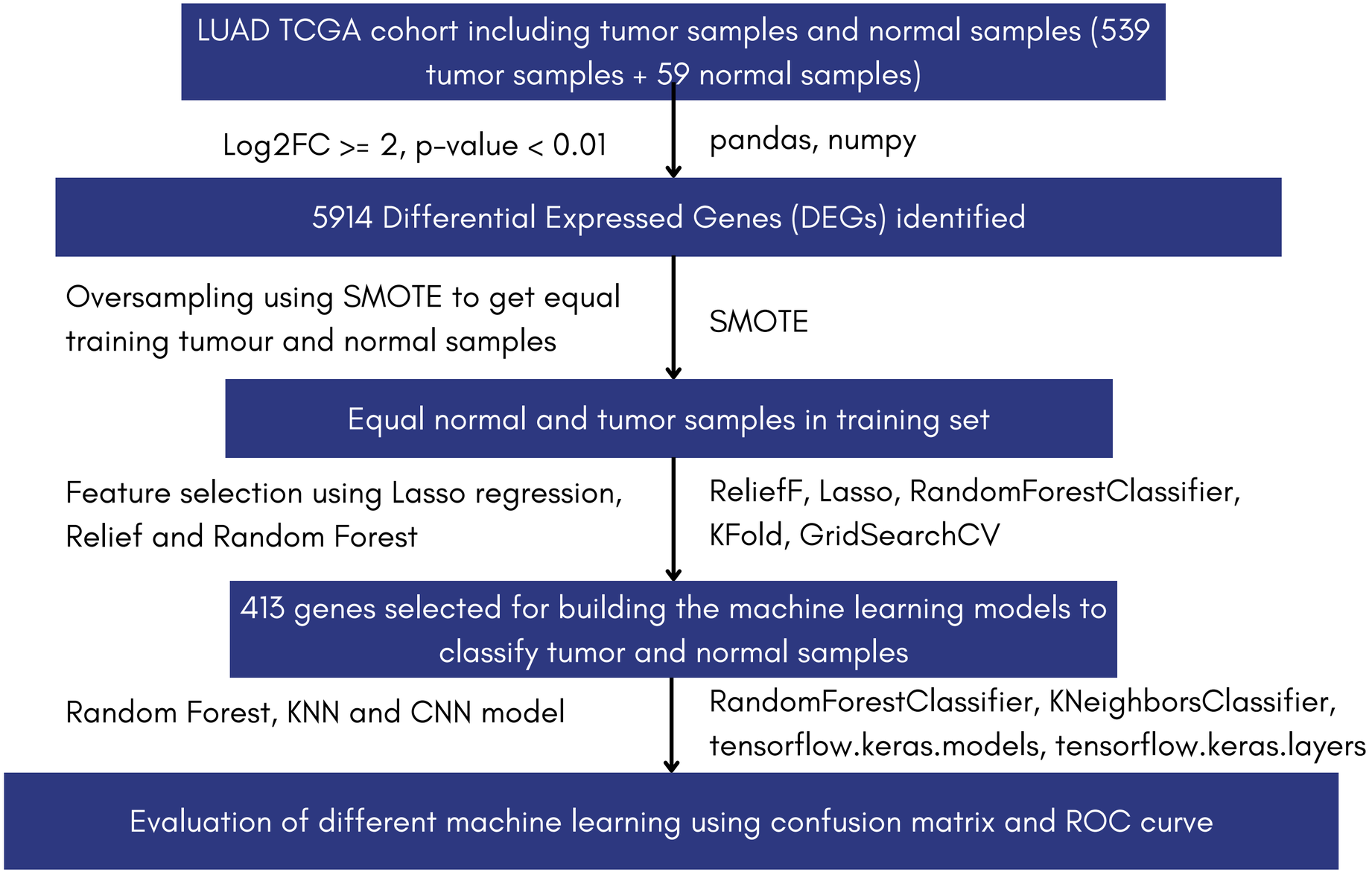


Aims

- 1) Use differential expressed genes (DEGs) between tumor samples and normal samples to diagnose lung adenocarcinoma, which is the most common type of lung cancer, by machine learning algorithms
- 2) Use differential expressed genes (DEGs) between early stage and late stage samples to build prognostic model for survival rate prediction



Workflow


LUAD TCGA cohort including tumor samples and normal samples (539 tumor samples + 59 normal samples)
Log2FC >= 2, p-value < 0.01
pandas, numpy
5914 Differential Expressed Genes (DEGs) identified
Oversampling using SMOTE to get equal training tumour and normal samples
SMOTE
Equal normal and tumor samples in training set
Feature selection using Lasso regression, Relief and Random Forest
ReliefF, Lasso, RandomForestClassifier, KFold, GridSearchCV
413 genes selected for building the machine learning models to classify tumor and normal samples
Random Forest, KNN and CNN model
RandomForestClassifier, KNeighborsClassifier, tensorflow.keras.models, tensorflow.keras.layers
Evaluation of different machine learning using confusion matrix and ROC curve

DEGs between normal and tumour samples

```
# Import result bw normal and tumour tissue
resLFC_LUAD = pd.read_csv("resLFC_LUAD.csv",index_col=0)
resLFC_LUAD.head()
```

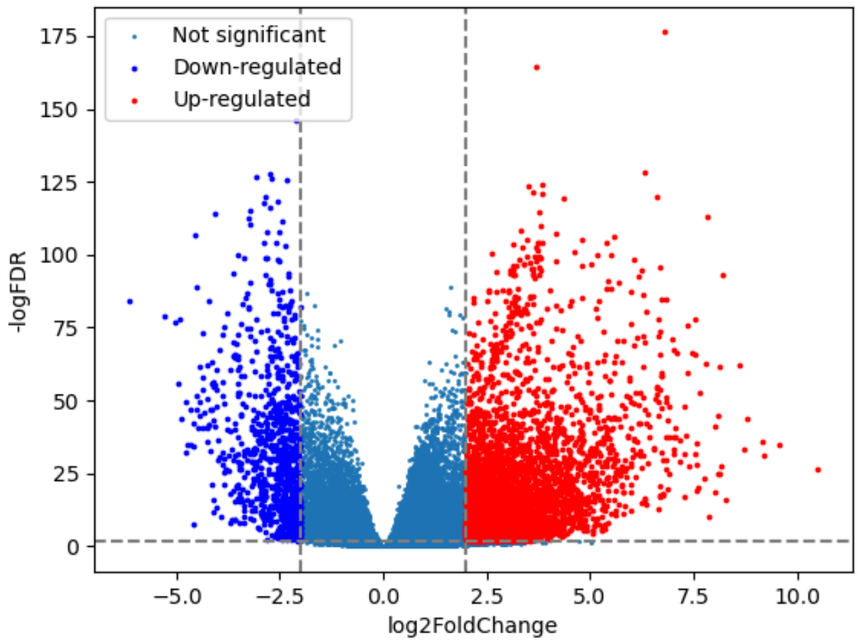
gene_id	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
ENSG00000000003.15	3043.417527	1.105787	0.102394	10.799368	3.465825e-27	5.997817e-26
ENSG00000000005.6	6.698799	1.580134	0.467129	3.382653	7.178931e-04	1.566460e-03
ENSG00000000419.13	1519.905188	0.253888	0.076538	3.317149	9.094119e-04	1.951526e-03
ENSG00000000457.14	752.504754	0.477561	0.065399	7.302215	2.830688e-13	1.786317e-12
ENSG00000000460.17	371.355110	1.737929	0.099519	17.463258	2.729069e-68	3.622678e-66

```
plt.scatter(x=resLFC_LUAD['log2FoldChange'],y=resLFC_LUAD['padj'].apply(lambda x:-np.log10(x)),s=1,label="Not significant")

# highlight down- or up- regulated genes
down = resLFC_LUAD[(resLFC_LUAD['log2FoldChange']<=-2)&(resLFC_LUAD['padj']<=0.01)]
up = resLFC_LUAD[(resLFC_LUAD['log2FoldChange']>=2)&(resLFC_LUAD['padj']<=0.01)]

plt.scatter(x=down['log2FoldChange'],y=down['padj'].apply(lambda x:-np.log10(x)),s=3,label="Down-regulated",color="blue")
plt.scatter(x=up['log2FoldChange'],y=up['padj'].apply(lambda x:-np.log10(x)),s=3,label="Up-regulated",color="red")

plt.xlabel("log2FoldChange")
plt.ylabel("-logFDR")
plt.axvline(-2,color="grey",linestyle="--")
plt.axvline(2,color="grey",linestyle="--")
plt.axhline(2,color="grey",linestyle="--")
plt.legend()
plt.show()
```



Input data

Gene expression data

```
# Import data set: gene expression and code
# gene expression
gene_expression_data = pd.read_csv("normalized_counts_LUAD.csv")
gene_expression_data
```

✓ 0.2s

	gene_id	TCGA-73-4658-01A-01R-1755-07	TCGA-44-2661-11A-01R-1758-07	TCGA-55-6986-11A-01R-1949-07	TCGA-55-8615-01A-11R-2403-07	TCGA-97-8177-01A-11R-2287-07	TCGA-49-6744-11A-01R-1858-07	TCGA-67-3771-01A-01R-0946-07	TCGA-49-6744-01A-11R-1858-07	TCGA-MP-A4SW-01A-21R-A24X-07	...
0	ENSG00000000938.13	2038.309881	3093.938452	2560.751728	686.023965	1450.396551	3694.950081	1114.561336	1428.059800	1356.044295	...
1	ENSG00000001084.13	1009.861734	924.844468	1659.551981	2221.782159	1705.903556	1062.349486	16094.413968	1324.236522	1754.117285	...
2	ENSG00000002079.14	40.890107	3.813792	9.978319	8.909402	6.939696	5.476028	12.356556	3.917860	12.115265	...
3	ENSG00000002726.21	66.911084	61.974114	21.006987	8285.743993	359.602451	30.118156	1147.924037	1474.094650	345.285050	...
4	ENSG00000002746.15	95.410250	63.881010	10.503494	41.205985	133.746876	41.070212	384.288886	57.788428	20.769026	...
...	...	...	...	...	...	...	...	...	...	...	...
5909	ENSG00000288571.1	0.000000	0.000000	0.000000	6.682052	1.892644	0.000000	25.948767	0.000000	0.000000	...
5910	ENSG00000288581.1	0.000000	2.860344	18.906288	0.000000	0.000000	1.369007	0.000000	0.000000	3.461504	...
5911	ENSG00000288611.1	7.434565	8.581031	15.230066	4.454701	1.261763	8.214042	9.885245	11.753579	6.923009	...
5912	ENSG00000288637.1	3.717282	0.953448	1.575524	0.000000	1.261763	0.000000	2.471311	0.000000	7.788385	...
5913	ENSG00000288638.1	121.431227	0.000000	0.000000	0.000000	153.935084	0.000000	2.471311	85.213445	52.787940	...

5914 rows × 599 columns

transform data

```
# Transform gene expression data: gene id as colnames and patients ID as rownames
transformed_gene_data = gene_expression_data.set_index("gene_id").transpose()
transformed_gene_data.index.name = "Patient_ID"
transformed_gene_data.reset_index(inplace=False)
transformed_gene_data
```

✓ 0.0s

gene_id Patient_ID	ENSG00000000938.13	ENSG00000001084.13	ENSG00000002079.14	ENSG00000002726.21	ENSG00000002746.15	ENSG00000004776.13
TCGA-73-4658-01A-01R-1755-07	2038.309881	1009.861734	40.890107	66.911084	95.410250	270.122525
TCGA-44-2661-11A-01R-1758-07	3093.938452	924.844468	3.813792	61.974114	63.881010	665.506638
TCGA-55-6986-11A-01R-1949-07	2560.751728	1659.551981	9.978319	21.006987	10.503494	397.032056
TCGA-55-8615-01A-11R-2403-07	686.023965	2221.782159	8.909402	8285.743993	41.205985	52.342738
TCGA-97-8177-01A-11R-2287-07	1450.396551	1705.903556	6.939696	359.602451	133.746876	97.786631

5 rows × 5914 columns

Clinical data

```
# code
tissue_class_data = pd.read_csv("LUAD_code.csv")
tissue_class_data
```

✓ 0.0s

	barcode	patient	shortLetterCode	days_to_last_follow_up	days_to_death
0	TCGA-73-4658-01A-01R-1755-07	TCGA-73-4658	TP	1600.0	1600.0
1	TCGA-44-2661-11A-01R-1758-07	TCGA-44-2661	NT	1159.0	NaN
2	TCGA-55-6986-11A-01R-1949-07	TCGA-55-6986	NT	3261.0	NaN
3	TCGA-55-8615-01A-11R-2403-07	TCGA-55-8615	TP	446.0	NaN
4	TCGA-97-8177-01A-11R-2287-07	TCGA-97-8177	TP	499.0	NaN
...	...	...	...	...	...
593	TCGA-64-1678-01A-01R-0946-07	TCGA-64-1678	TP	1189.0	NaN
594	TCGA-78-7155-01A-11R-2039-07	TCGA-78-7155	TP	NaN	1171.0
595	TCGA-78-7220-01A-11R-2039-07	TCGA-78-7220	TP	NaN	807.0
596	TCGA-80-5611-01A-01R-1628-07	TCGA-80-5611	TP	2595.0	NaN
597	TCGA-93-8067-01A-11R-2287-07	TCGA-93-8067	TP	186.0	NaN

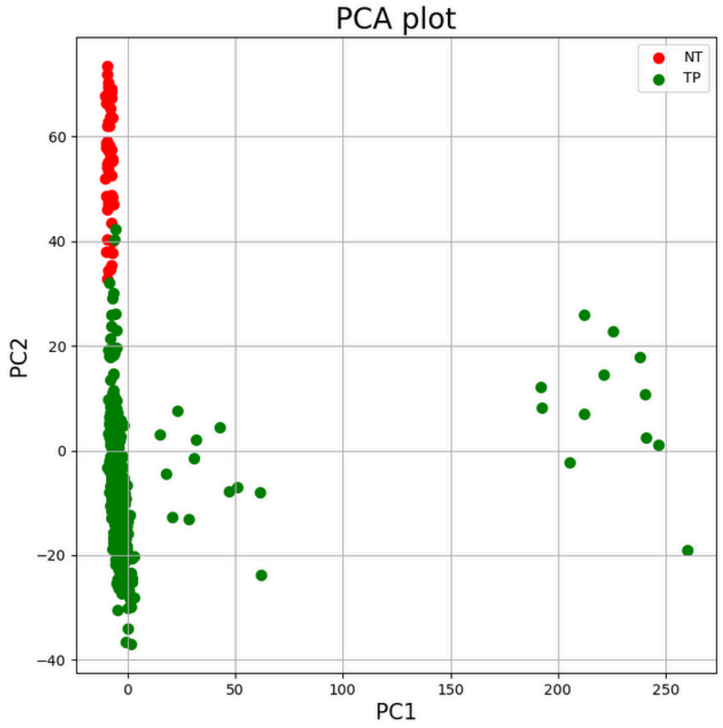
598 rows × 5 columns

PCA plot based on DEGs between tumor and normal samples

```
# 5) Scale data
scaler = StandardScaler()
X_scale = scaler.fit_transform(X)
# PCA plot
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_scale)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, tissue_class_data["shortLetterCode"]], axis = 1)
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('PC1', fontsize = 15)
ax.set_ylabel('PC2', fontsize = 15)
ax.set_title('PCA plot', fontsize = 20)

targets = ['NT', "TP"]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['shortLetterCode'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
plt.savefig("PCA_plot.png")
```

✓ 0.3s



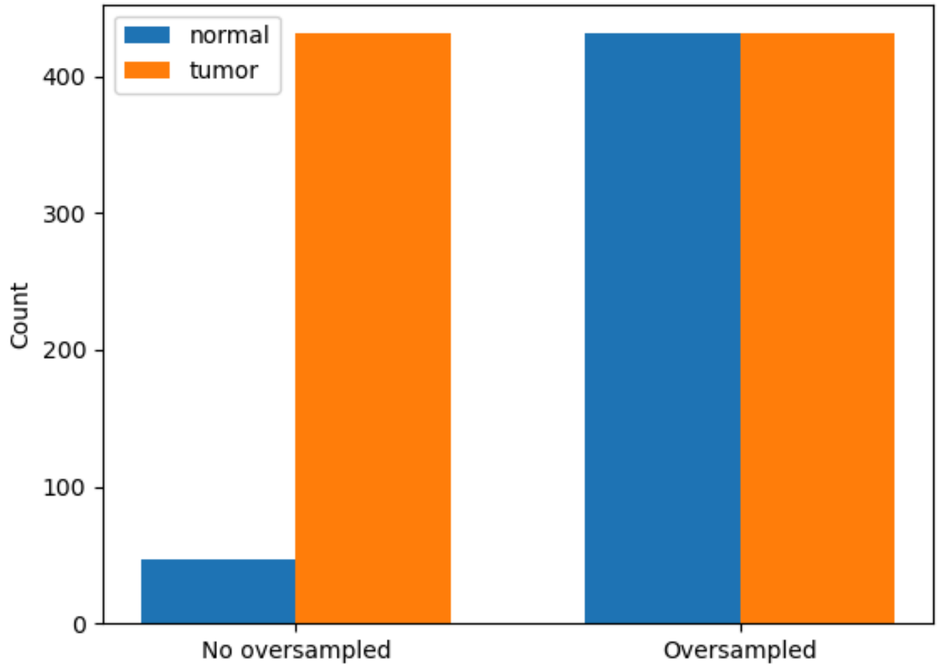
Oversampling using SMOTE to get equal sample size of training tumor and normal

```
tissue_class_data = pd.read_csv("LUAD_code.csv")

tissue_class_data['Class'] = np.where(tissue_class_data['shortLetterCode'] == 'TP', 1, 0)
tissue_class_data
```

	barcode	patient	shortLetterCode	days_to_last_follow_up	days_to_death	Class
0	TCGA-73-4658-01A-01R-1755-07	TCGA-73-4658	TP	1600.0	1600.0	1
1	TCGA-44-2661-11A-01R-1758-07	TCGA-44-2661	NT	1159.0	NaN	0
2	TCGA-55-6986-11A-01R-1949-07	TCGA-55-6986	NT	3261.0	NaN	0
3	TCGA-55-8615-01A-11R-2403-07	TCGA-55-8615	TP	446.0	NaN	1
4	TCGA-97-8177-01A-11R-2287-07	TCGA-97-8177	TP	499.0	NaN	1
...	...	...	...	...	...	...
593	TCGA-64-1678-01A-01R-0946-07	TCGA-64-1678	TP	1189.0	NaN	1
594	TCGA-78-7155-01A-11R-2039-07	TCGA-78-7155	TP	NaN	1171.0	1
595	TCGA-78-7220-01A-11R-2039-07	TCGA-78-7220	TP	NaN	807.0	1
596	TCGA-80-5611-01A-01R-1628-07	TCGA-80-5611	TP	2595.0	NaN	1
597	TCGA-93-8067-01A-11R-2287-07	TCGA-93-8067	TP	186.0	NaN	1

598 rows × 6 columns




```
# Assign gene expression data and class data to X and y
X = transformed_gene_data.values
y = tissue_class_data_1.values

# Split the data into test set and training set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)
print("Shape of Train Features: {}".format(X_train.shape))
print("Shape of Test Features: {}".format(X_test.shape))
print("Shape of Train Target: {}".format(y_train.shape))
print("Shape of Test Target: {}".format(y_test.shape))
```

```
Shape of Train Features: (478, 5914)
Shape of Test Features: (120, 5914)
Shape of Train Target: (478, 1)
Shape of Test Target: (120, 1)
```

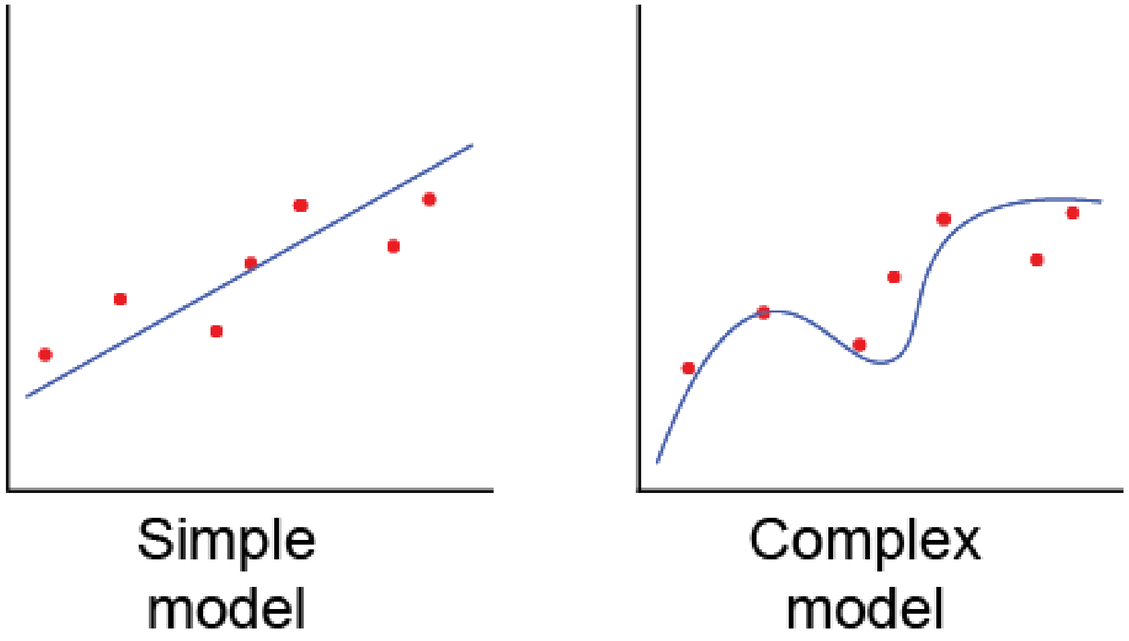
```
# Oversampling
smote = SMOTE(random_state=42)
X_train_oversampled, y_train_oversampled = smote.fit_resample(X_train, y_train) # only apply SMOTE on training data so it will not affect the test data
```

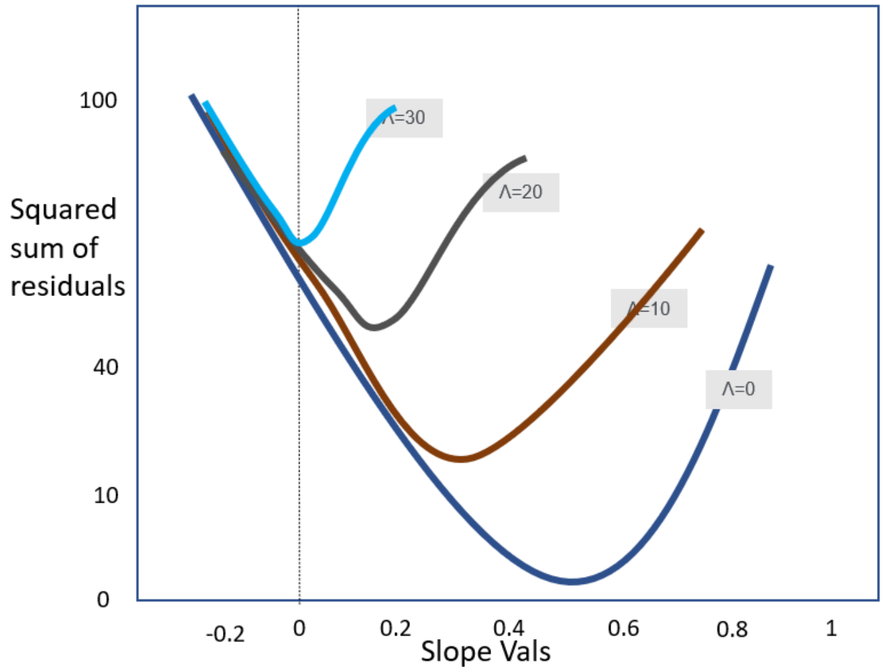
Feature Selection

1. Lasso
2. Relief
3. Random Forest

Feature Selection - Lasso Regression

- We can only estimate linear regression if the number of features are lower than the number of data points
- Lasso regression – shrink the less importance features coefficient to zero --> can be used for feature selection





Feature Selection - Lasso Regression

Choose the best lambda for lasso regression

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_oversampled)
X_test_scaled = scaler.transform(X_test)
```

```
# range of alpha values
params = {'alpha': np.logspace(-4, 1, 100)}
# 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)
# Lasso model
lasso = Lasso(max_iter=50000)

# GridSearchCV --> best alpha
lasso_cv = GridSearchCV(lasso, param_grid=params, cv=kf, n_jobs=-1)
lasso_cv.fit(X_train_scaled, y_train_oversampled)

# Print best alpha
best_alpha = lasso_cv.best_params_['alpha']
print(f"Best Alpha: {best_alpha}")
```

```
Best Alpha: 0.001023531021899027
```

```
# Fit the Lasso model with the best alpha
lasso_best = Lasso(alpha= best_alpha, max_iter=50000)
lasso_best.fit(X_train_scaled, y_train_oversampled)
```

```
Lasso
Lasso(alpha=0.001023531021899027, max_iter=50000)
```

Choose most importance feature using non-zero coefficient genes

```
# Using np.abs() to make coefficients positive.
lasso1_coef = np.abs(lasso_best.coef_)
threshold = 0.01  # Threshold for lasso coeficient
names=transformed_gene_data.columns # names of genes
feature_subset=np.array(names)[lasso1_coef>0.001]
print(len(feature_subset))
```

✓ 0.0s

```
200
```

```
# Change np array to panda table
selected_genes_lasso = pd.DataFrame(feature_subset, columns=['gene_id'])
selected_genes_lasso
```

	gene_id
0	ENSG00000007402.12
1	ENSG00000011201.12
2	ENSG00000034971.17
3	ENSG00000064195.7
4	ENSG00000065618.21
...	...
195	ENSG00000287035.1
196	ENSG00000287641.1
197	ENSG00000287652.1
198	ENSG00000287784.1
199	ENSG00000287879.1

200 rows × 1 columns

Feature Selection - Relief

Relief feature selection that used in machine learning is to assess the importance of features (attributes) based on their ability to differentiate between data instances (samples).

Two main points of relief algorithm:

1. Nearby instances
2. Differentiation between classes

Relief help to identify features that are most useful for distinguishing between classes.

Ex: Cat VS Dog

- Number of legs --> not a useful feature
- Barking sound level --> important feature

Feature Selection - Relief

```python
from skrebate import ReliefF
import numpy as np
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
import pandas as pd

# Function to select features using ReliefF
def select_features_relief(X, y, num_features, feature_names):
    # Standardize features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply ReliefF feature selection
    relieff = ReliefF(n_features_to_select=num_features, n_neighbors=10)
    relieff.fit(X_scaled, y)

    # Get feature importance scores
    feature_scores = relieff.feature_importances_

    # Select top features
    selected_feature_indices = np.argsort(feature_scores)[::-1][:num_features]
    selected_features = [feature_names[i] for i in selected_feature_indices]  # Map indices to gene IDs

    return selected_features
```

```python
# Select features and output their IDs
num_features_to_select = 200  # Adjust this to the desired number of features
selected_gene_ids = select_features_relief(X_resampled, y_resampled, num_features_to_select, gene_ids)

print(f"Top {num_features_to_select} selected gene IDs:")
selected_gene_ids
```

ReliefFSelector constructor has only 2 parameters:

1. n_neighbors -> the number of neighbors to consider when assigning feature importance scores.

2. n_features -> it represents the number of top features (according to the relieff score) to retain after feature selection is applied.

```
Top 200 selected gene IDs:
[1]: ['ENSG00000171885.18',
      'ENSG00000261863.1',
      'ENSG00000138821.13',
      'ENSG00000078081.8',
      'ENSG00000122852.15',
      'ENSG00000123119.12',
      'ENSG00000168484.12',
      'ENSG00000274736.5',
```

Machine Learning - Random Forest

Random Forest:

- Random Forest builds multiple decision trees, and during the construction of these trees, features are split at various nodes to reduce impurity.
- The importance of a feature is determined by how much it contributes to reducing impurity across all the trees in the forest.

Steps:

1. Tunes Random Forest hyperparameters with cross-validation to optimize performance.
2. Trains the best Random Forest model on the balanced training set.
3. Evaluates its performance

Feature Selection - Random Forest

```python
# Random forest
param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
}
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
# Best parameters from grid search
best_params = grid_search.best_params_
print("Best parameters:", best_params)
# Train Random Forest Classifier with Best Parameters
rf_best = grid_search.best_estimator_
rf_best.fit(X_train_resampled, y_train_resampled)
# Evaluate Model Performance
y_pred = rf_best.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        18
           1       0.99      0.99      0.99       162

    accuracy                           0.99       180
   macro avg       0.97      0.97      0.97       180
weighted avg       0.99      0.99      0.99       180
```

Machine Learning

1. KNN
2. CNN
3. Random Forest

Machine Learning - KNN

```python
import pandas as pd

# Load the combined genes data
genes_data = pd.read_csv('combined_data.csv')

# Load the patient data
patient_data = pd.read_csv('normalized_counts_LUAD.csv')

# Merge the two datasets on 'Gene ID'
merged_data = pd.merge(genes_data, patient_data, on='gene_id', how='inner')  # Use 'inner' for common IDs only

# Change the row and column
transformed_gene_data = merged_data.set_index("gene_id").transpose()
transformed_gene_data.index.name = "Patient_ID"
transformed_gene_data.reset_index(inplace=False)
transformed_gene_data
```

gene_id	ENSG00000007402.12	ENSG00000011201.12	ENSG00000034971.17	ENSG00000064195.7	ENSG00000065618.21	ENSG00000079462.8	EN
Patient_ID							
TCGA-73-4658-01A-01R-1755-07	257.731584	1812.794745	9.912753	307.295350	16762.465693	852.496777	
TCGA-44-2661-11A-01R-1758-07	11761.733357	13258.646568	150.644769	62.927562	151.598217	348.961933	
TCGA-55-6986-11A-01R-1949-07	7409.164353	9795.558088	110.811857	33.611179	87.178996	198.516028	
TCGA-55-8615-01A-11R-2403-07	3961.342928	126.958981	1.113675	161.482914	986.716287	5004.856654	
TCGA-97-8177-01A-11R-2287-07	1960.148797	1864.885692	1.261763	403.764155	2902.685747	1186.057206	
...	...	...	...	...	...	...	
TCGA-64-1678-01A-01R-0946-07	579.108861	411.631386	0.000000	34.302615	38.338217	5020.288667	
TCGA-78-7155-01A-11R-2039-07	32.364181	3080.821052	1.244776	1804.925465	13.692538	1304.525439	
TCGA-78-7220-01A-11R-2039-07	208.746183	258.727664	1.960058	112.703338	386.131438	1059.411381	
TCGA-80-5611-01A-01R-1628-07	19.563442	401.625948	0.000000	34.523720	43.730046	2006.978927	
TCGA-93-8067-01A-11R-2287-07	492.759149	524.237620	4.237487	147.706674	905.006053	974.016548	

Machine Learning - KNN

```
import numpy as np

# class data
tissue_class_data = pd.read_csv("LUAD_code.csv")

tissue_class_data['Class'] = np.where(tissue_class_data['shortLetterCode'] == 'TP', 1, 0)
tissue_class_data
```

	barcode	patient	shortLetterCode	days_to_last_follow_up	days_to_death	Class
0	TCGA-73-4658-01A-01R-1755-07	TCGA-73-4658	TP	1600.0	1600.0	1
1	TCGA-44-2661-11A-01R-1758-07	TCGA-44-2661	NT	1159.0	NaN	0
2	TCGA-55-6986-11A-01R-1949-07	TCGA-55-6986	NT	3261.0	NaN	0
3	TCGA-55-8615-01A-11R-2403-07	TCGA-55-8615	TP	446.0	NaN	1
4	TCGA-97-8177-01A-11R-2287-07	TCGA-97-8177	TP	499.0	NaN	1
...	...	...	...	...	...	...
593	TCGA-64-1678-01A-01R-0946-07	TCGA-64-1678	TP	1189.0	NaN	1
594	TCGA-78-7155-01A-11R-2039-07	TCGA-78-7155	TP	NaN	1171.0	1
595	TCGA-78-7220-01A-11R-2039-07	TCGA-78-7220	TP	NaN	807.0	1
596	TCGA-80-5611-01A-01R-1628-07	TCGA-80-5611	TP	2595.0	NaN	1
597	TCGA-93-8067-01A-11R-2287-07	TCGA-93-8067	TP	186.0	NaN	1

598 rows × 6 columns

Machine Learning - KNN

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score
from imblearn.over_sampling import SMOTE


# Separate features (X) and target variable (y)
# Replace 'Target' with the actual name of your target column
X = transformed_gene_data.values
y = np.asarray(tissue_class_data["Class"])


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE for oversampling
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```python
# KNN model
# Normalize the feature data for KNN
scaler = StandardScaler()
X_train_resampled = scaler.fit_transform(X_train_resampled)
X_test = scaler.transform(X_test)

# Train the KNN model
knn = KNeighborsClassifier(n_neighbors=5)  # Adjust n_neighbors as needed
knn.fit(X_train_resampled, y_train_resampled)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
#print("Classification Report:")
#print(classification_report(y_test, y_pred))
```

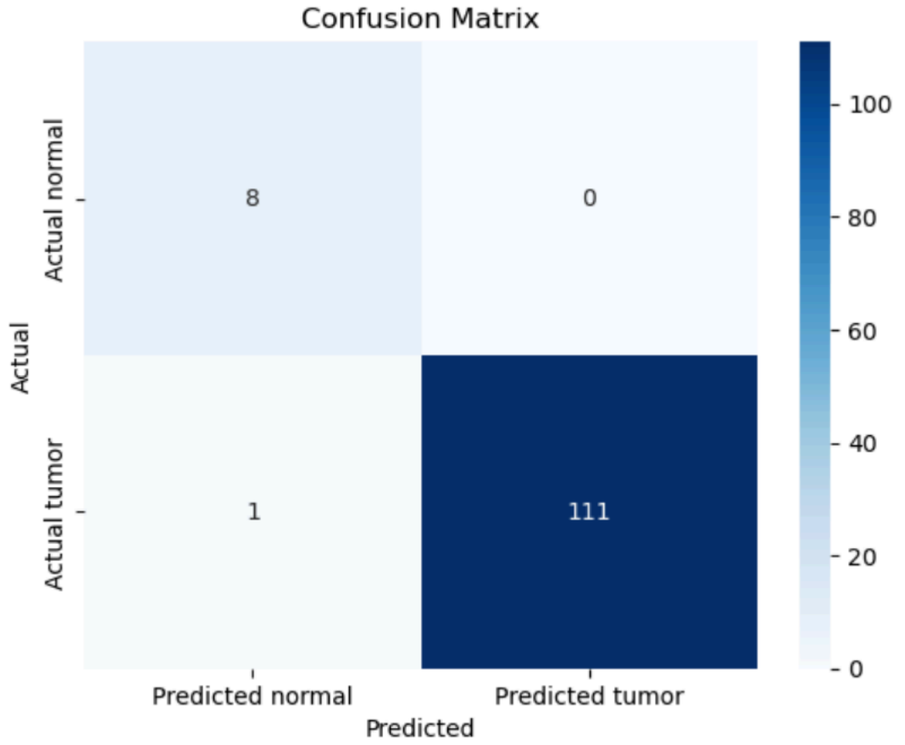
```
Accuracy: 0.9916666666666667
```

Machine Learning - KNN

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute and plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=knn.classes_)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted normal', 'Predicted tumor'],
            yticklabels=['Actual normal', 'Actual tumor'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

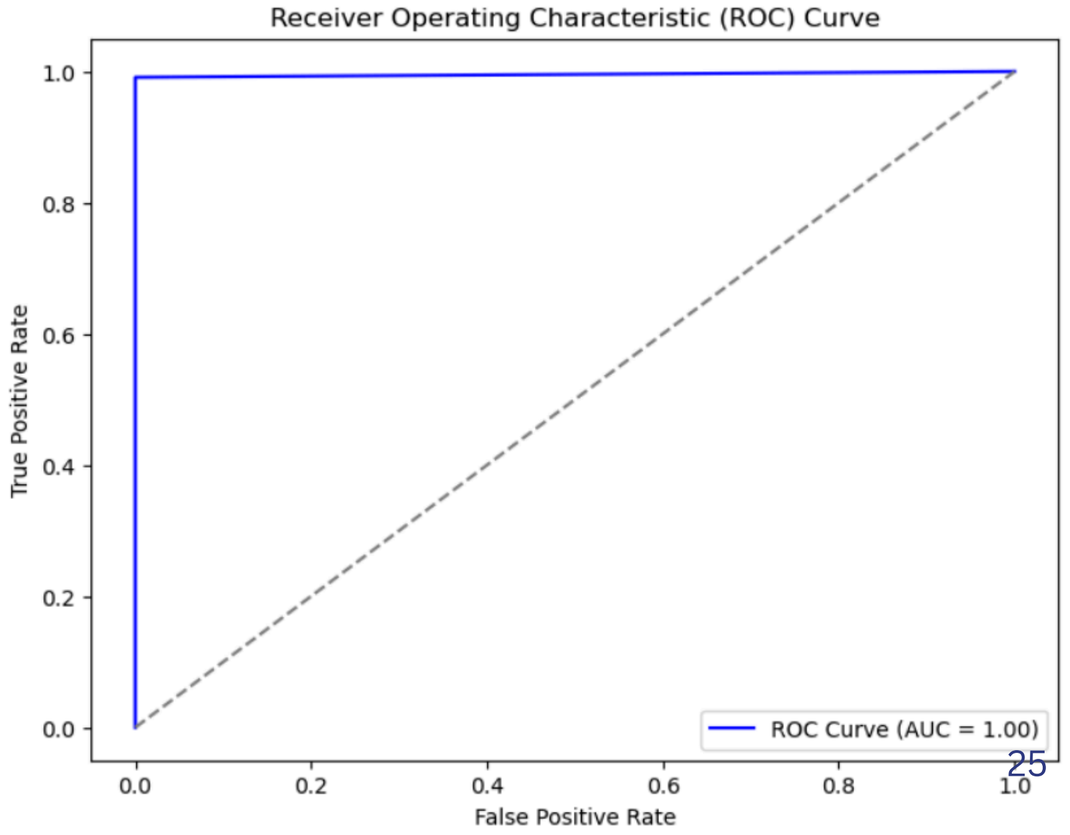


Machine Learning - KNN

```python
from sklearn.metrics import roc_curve, auc

# Compute ROC curve and AUC for KNN
y_pred_proba_binary = knn.predict_proba(X_test)[:, 1]  # Use the positive class probabilities
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_binary)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



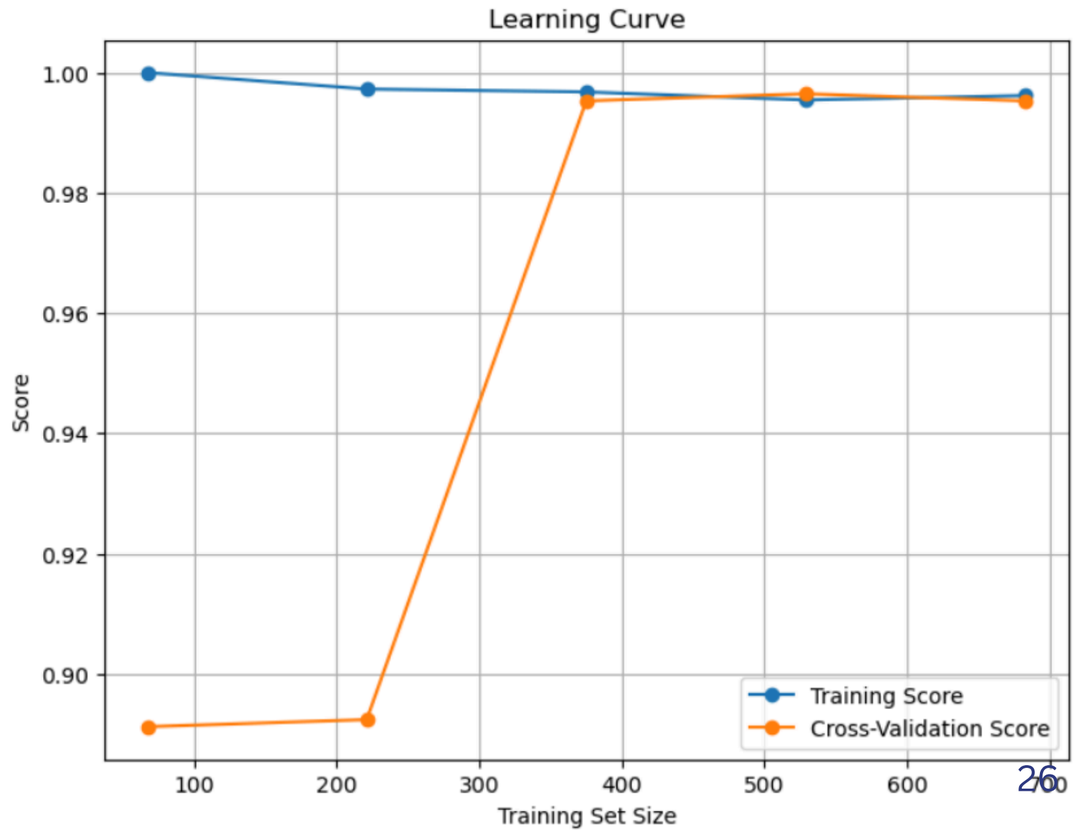
Machine Learning - KNN

```python
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(knn, X_train_resampled, y_train_resampled, cv=5)

train_scores_mean = train_scores.mean(axis=1)
test_scores_mean = test_scores.mean(axis=1)

plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_scores_mean, label="Training Score", marker='o')
plt.plot(train_sizes, test_scores_mean, label="Cross-Validation Score", marker='o')
plt.title("Learning Curve")
plt.xlabel("Training Set Size")
plt.ylabel("Score")
plt.legend()
plt.grid()
plt.show()
```



Machine Learning - CNN

Reshape the data to 2D shape and build the CNN model

```
# 02. CNN
# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_oversampled)
X_test_scaled = scaler.transform(X_test)

# Reshaping the data to 2D
X_train_cnn = X_train_scaled.reshape(X_train_scaled.shape[0], X_train_scaled.shape[1], 1)
X_test_cnn = X_test_scaled.reshape(X_test_scaled.shape[0], X_test_scaled.shape[1], 1)
```

```
# Building the CNN model
model = Sequential([
    Reshape((X_train_scaled.shape[1], 1, 1), input_shape=(X_train_scaled.shape[1], 1)),
    Conv2D(32, kernel_size=(1, 1), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

Compile model

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
reshape_3 (Reshape)	(None, 413, 1, 1)	0
conv2d_3 (Conv2D)	(None, 413, 1, 32)	64
flatten_3 (Flatten)	(None, 13216)	0
dense_6 (Dense)	(None, 64)	845,888
dense_7 (Dense)	(None, 1)	65

Total params: 846,017 (3.23 MB)

Trainable params: 846,017 (3.23 MB)

Non-trainable params: 0 (0.00 B)

Machine Learning - CNN

Fit model on training data

```
history = model.fit(X_train_cnn, y_train_oversampled, epochs=50, batch_size=32, validation_split=0.3)
```

```
Epoch 1/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.8890 - loss: 0.1454 - val_accuracy: 1.0000 - val_loss: 7.8004e-06
Epoch 2/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9953 - loss: 0.0161 - val_accuracy: 1.0000 - val_loss: 4.7660e-04
Epoch 3/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 3.2657e-04 - val_accuracy: 1.0000 - val_loss: 1.9314e-04
Epoch 4/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 1.0000 - loss: 1.9134e-04 - val_accuracy: 1.0000 - val_loss: 8.0271e-05
Epoch 5/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 4.2287e-05 - val_accuracy: 1.0000 - val_loss: 5.5620e-05
Epoch 6/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 5.5986e-05 - val_accuracy: 1.0000 - val_loss: 4.1408e-05
Epoch 7/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 2.4868e-05 - val_accuracy: 1.0000 - val_loss: 3.4888e-05
Epoch 8/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 3.4749e-05 - val_accuracy: 1.0000 - val_loss: 2.9196e-05
Epoch 9/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 2.5715e-05 - val_accuracy: 1.0000 - val_loss: 2.4800e-05
Epoch 10/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 1.5949e-05 - val_accuracy: 1.0000 - val_loss: 2.2134e-05
Epoch 11/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 1.5585e-05 - val_accuracy: 1.0000 - val_loss: 2.0518e-05
Epoch 12/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 8.3645e-06 - val_accuracy: 1.0000 - val_loss: 1.8497e-05
Epoch 13/50
...
Epoch 49/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 1.0000 - loss: 4.2349e-06 - val_accuracy: 1.0000 - val_loss: 3.2625e-06
Epoch 50/50
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 1.0000 - loss: 4.9612e-06 - val_accuracy: 1.0000 - val_loss: 3.1446e-06
```

Machine Learning - CNN

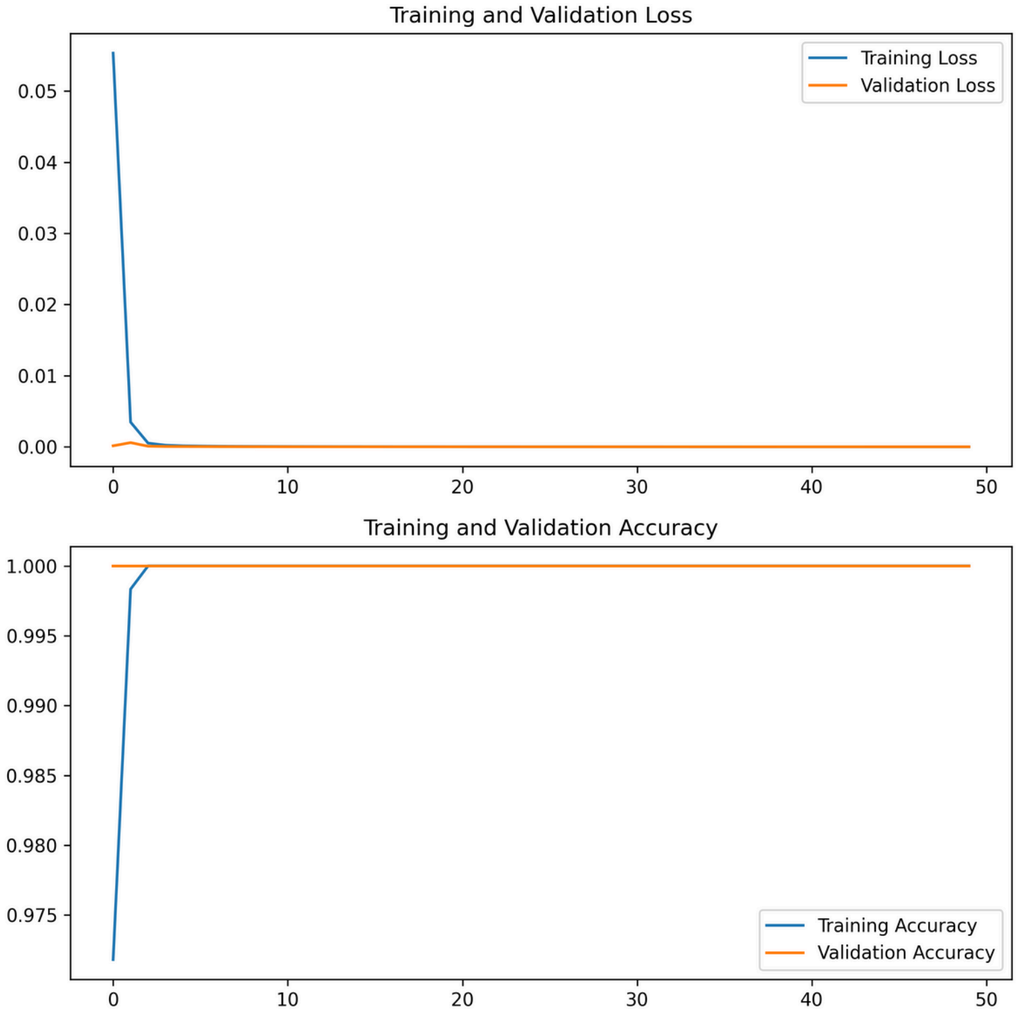
Training history

```
import matplotlib.pyplot as plt
# Training Loss
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

# Training Accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 2)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
```



Machine Learning - CNN

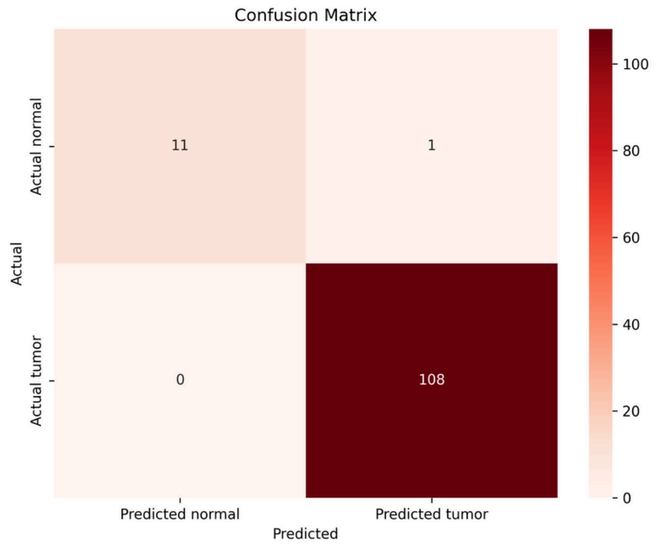
```python
# Confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

y_pred = model.predict(X_test_cnn)
y_pred_classes = (y_pred > 0.5).astype(int)

cm = confusion_matrix(y_test, y_pred_classes)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds',
            xticklabels=['Predicted normal', 'Predicted tumor'],
            yticklabels=['Actual normal', 'Actual tumor'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```



$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Test data evaluation

```python
# Evaluate the model
loss, accuracy = model.evaluate(X_test_cnn, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9946 - loss: 0.0497
Test Loss: 0.07641054689884186
Test Accuracy: 0.9916666746139526
```

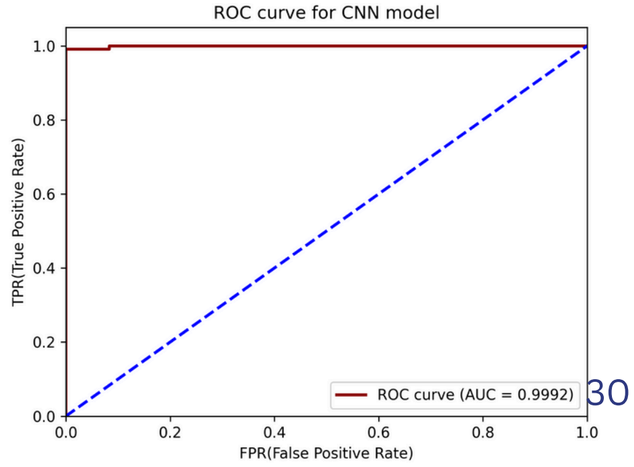
```python
from sklearn.metrics import roc_curve, auc, roc_auc_score

# y_predict probability
y_pred_prob = model.predict(X_test_cnn)

# Compute ROC curve and ROC area for each class
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkred', lw=lw, label='ROC curve (AUC = %0.4f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='blue', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('FPR(False Positive Rate)')
plt.ylabel('TPR(True Positive Rate)')
plt.title('ROC curve for CNN model')
plt.legend(loc="lower right")
plt.show()
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



Machine Learning - Random Forest

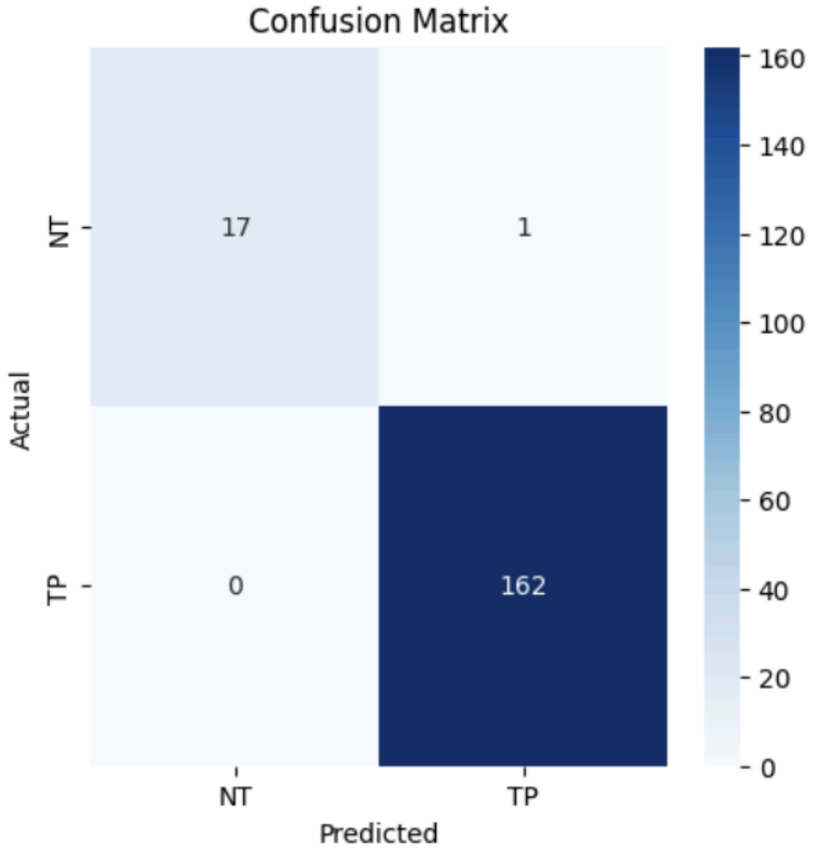
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
# Train Random Forest Classifier
param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
}
rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_resampled, y_train_resampled)
# Evaluate the best model
rf_best = grid_search.best_estimator_
y_pred = rf_best.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

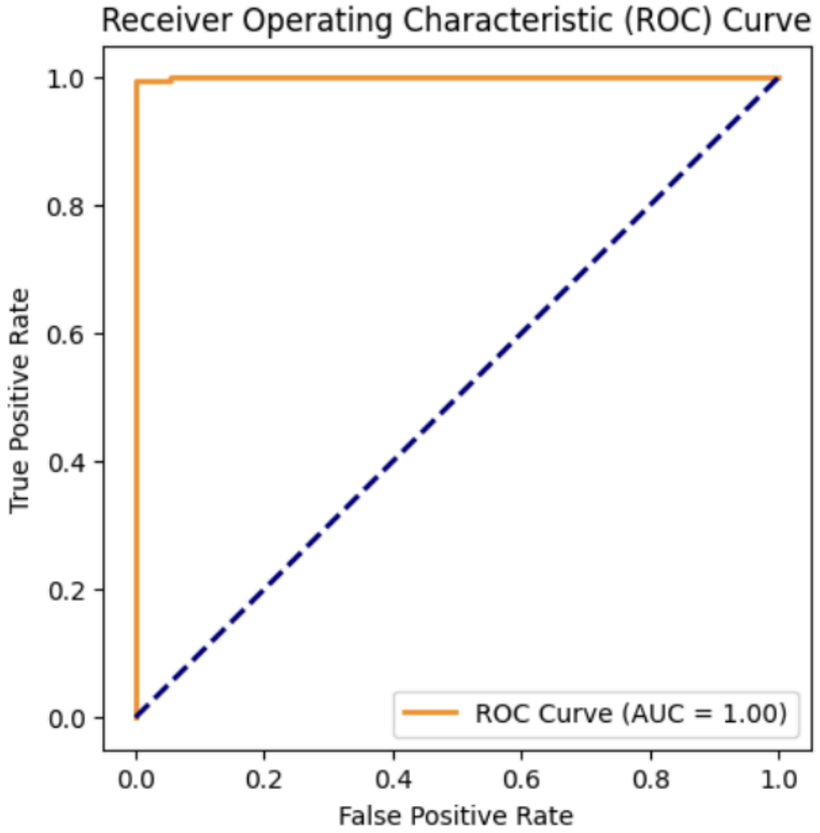
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        18
           1       0.99      1.00      1.00       162

    accuracy                           0.99       180
   macro avg       1.00      0.97      0.98       180
weighted avg       0.99      0.99      0.99       180
```

Feature Selection - Random Forest





Predicting the Prognosis of Patients Using LASSO Cox Regression

1. Data Preparation
2. Feature Selection with LASSO
3. Cox Proportional Hazards Model
4. Prediction

Prognosis Prediction

Data Preparation

```
# Load files
patients_df = pd.read_csv('LUAD_code.csv')
gene_expression_df = pd.read_csv('normalized_counts_stage.csv')
differential_expression_df = pd.read_csv('resLFC_LUAD_filt_stage.csv')

gene_expression_long = pd.melt(gene_expression_df, id_vars=['gene_id'], var_name='patient', value_name='expression')
merged_genes_df = pd.merge(gene_expression_long, differential_expression_df, on='gene_id', how='inner')

# change patient into barcode in LUAD_code.csv
merged_genes_df = merged_genes_df.rename(columns={'patient': 'barcode'})
final_df = pd.merge(patients_df, merged_genes_df, on='barcode', how='inner')

# Define survival time and status
final_df['survival_time'] = final_df[['days_to_last_follow_up', 'days_to_death']].min(axis=1)
final_df['status'] = final_df['days_to_death'].apply(lambda x: 1 if pd.notna(x) else 0)

wide_df = final_df.pivot_table(index=['patient', 'survival_time', 'status'], columns='gene_id', values='expression').reset_index()
```

Prognosis Prediction

Data Preparation

```
# define feature matrix X
X = wide_df.drop(columns=['patient', 'survival_time', 'status'])

# Survival time
y_time = wide_df['survival_time']

# Present statue (still alive or not)
y_event = wide_df['status']

# Set data into train and test
X_train, X_test, y_train_time, y_test_time, y_train_event, y_test_event = train_test_split(
    X, y_time, y_event, test_size=0.2, random_state=42)
```

Prognosis Prediction

Feature Selection Using LASSO

```
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply LASSO for feature selection
lasso = LassoCV(cv=5, max_iter=100000, alphas=[0.001, 0.01, 0.1, 1.0, 10.0, 20.0, 30.0, 50.0])
lasso.fit(X_train_scaled, y_train_time)

print("Best alpha:", lasso.alpha_)

selected_genes = X.columns[lasso.coef_ != 0]
print(f"# of genes selected: {len(selected_genes)}")
print(f"Selected genes: {selected_genes}")
```

Prognosis Prediction

Cox Regression

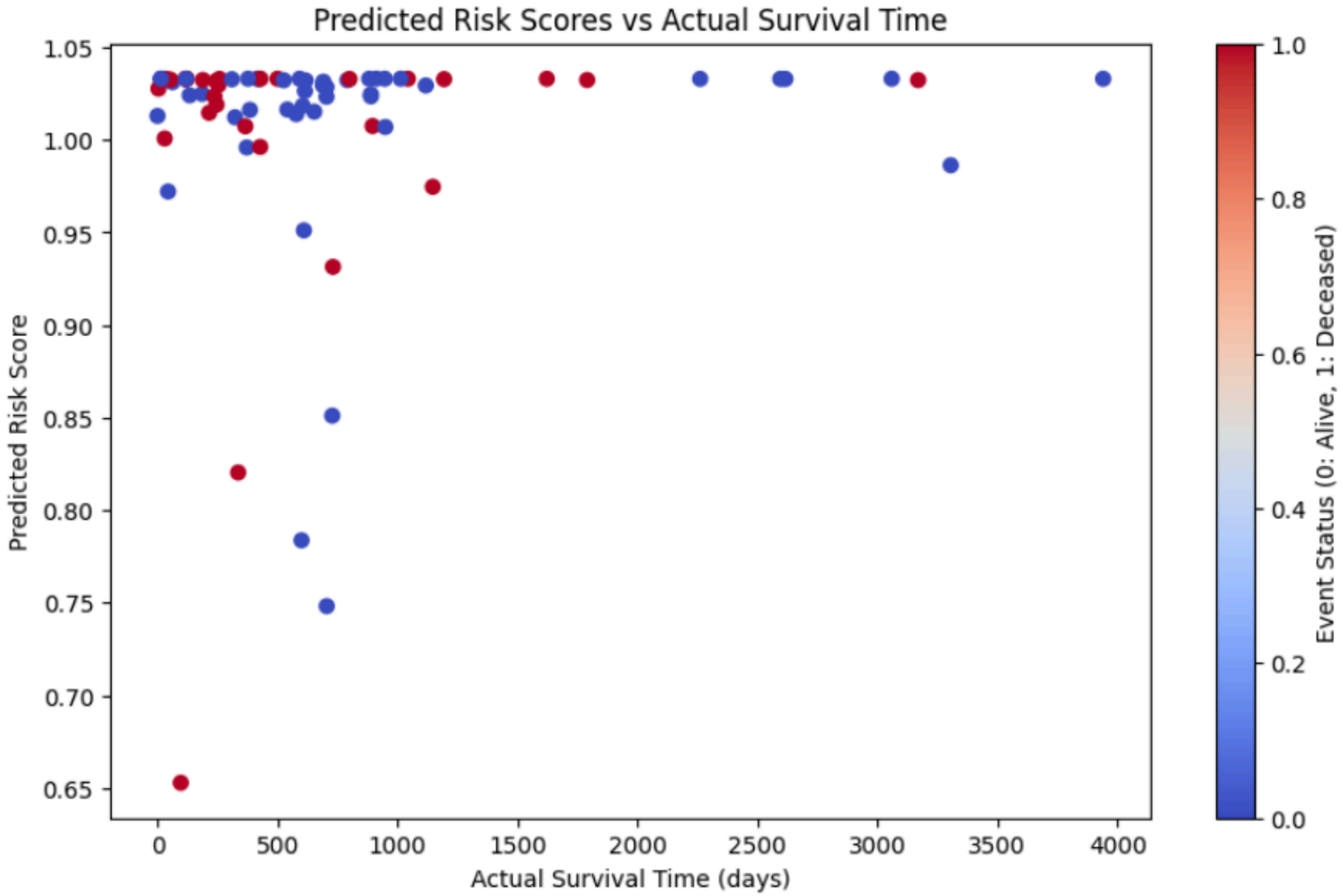
```
# COX regression
X_train_selected = X_train[selected_genes]
X_test_selected = X_test[selected_genes]

cox_df_train = pd.concat([pd.DataFrame({'survival_time': y_train_time, 'status': y_train_event}),
                          pd.DataFrame(X_train_selected)], axis=1)

cox_model = CoxPHFitter()
cox_model.fit(cox_df_train,  duration_col='survival_time', event_col='status')
cox_model.print_summary()
```

Prognosis Prediction

Test and Result



Prognosis Prediction

Potential Causes of Error

1. Specific genes are highly correlated but were not eliminated by Lasso.
2. Incomplete data.

Conclusion & Discussion

1. Use feature selection to reduce the complexity of models: from 5914 genes to 413 genes
2. Using machine learning models: CNN, KNN and Random Forest can achieve high accuracy of sample classification
3. Overall accuracy is 0.99, there is no significant difference between the models.

Problems & Takeaway Message

Problems:

1. Imbalance number of tumor and normal samples from input data
2. High number of features (genes) compared with number of patient samples

Takeaway message:

1. With the advance of AI and technology, we can finally use machine learning to differentiate between cancerous and non-cancerous genes, which may be helpful in the future.
2. When we encounter imbalance data, we can use oversampling to balance the data.

Workload Distribution

阮氏銀江 (Nguyen Thi Ngan Giang)

- Collect data, conceptualization
- Coding: DEGs identification, PCA cluster, lasso regression for classification feature selection, CNN model
- Present

陳亭安 (Annie)

- Coding – Data processing, Random Forest for feature selection and modeling
- Making and standardize the slides
- Present

方欣茹

- Parts of the coding (Relief feature selection, KNN Machine Learning)
- Presentation slides and presenter

李哲萱

- Do the prognosis prediction using LASSO Cox regression
- Present

Thank You

11 December, 2024