Sept. 5, 2023

AI6124 – Neuro Evolution and Fuzzy Intelligence
Assignment # 1

Reinelle Jan Bugnot
G2304329L

Import Data

Import provided Ovarian dataset from NUH. In this assignment, I used *g2c2Train.txt* and *g2c2Test.txt*.

```
data_train = pd.read_csv('Ovarian-NUH/g2c2Train.txt', skiprows=1, header=None, sep='\t')
print(f"Shape of training data: {data_train.shape}")

data_test = pd.read_csv('Ovarian-NUH/g2c2Test.txt', skiprows=1, header=None, sep='\t')
print(f"Shape of testing data: {data_test.shape}")
```

```
Shape of training data: (55, 29)
Shape of testing data: (53, 29)
```

```
data_train.head()
```

	0	1	2	3	4	5	6	7	8	9	...	19	20	21	22	23	24	25	26	27	28
0	16.0	355	0.35	122	8.0	11.0	5.5	44.0	2.7	98	...	9.2	2.10	0.59	28	343	0.7125	95	0.42	26.75	0
1	47.0	415	0.40	122	18.0	12.0	4.5	46.0	2.4	118	...	4.0	0.40	0.94	22	87	1.2065	119	0.96	7.19	0
2	72.0	270	0.36	122	6.0	9.0	2.5	49.0	5.6	97	...	5.1	0.70	1.54	358	695	1.1700	94	0.92	53.00	0
3	74.0	188	0.35	117	7.0	8.0	2.5	52.5	4.7	103	...	4.0	0.50	1.37	16	291	0.9100	90	0.75	8.90	1
4	48.0	320	0.40	140	8.5	9.0	2.5	52.0	3.6	132	...	4.4	0.62	0.80	19	363	1.0000	83	2.20	8.16	1

Process Data

To prepare the data for modelling, I simply separated the training features from the target variables for both train and test data, and then used a scaling function from sklearn (StandardScaler()).

In [5]:
```
# Split dataset into training input features and target variable
X_train = data_train.iloc[:, :-1]
y_train = data_train.iloc[:, -1]

X_test= data_test.iloc[:, :-1]
y_test = data_test.iloc[:, -1]
```

In [6]:
```
# Data Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test= scaler.fit_transform(X_test)
```

Train Model

I used a simple support vector classifier model with the following parameters as a decision tool (I used a simple grid search algorithm to quickly find optimal hyperparameters).

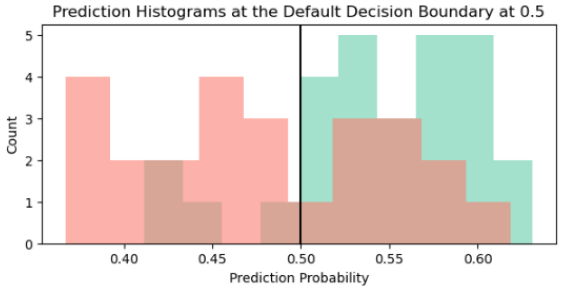
In [8]:
```
# Build Model
model = SVC(C=grid.best_params_['C'], gamma=grid.best_params_['gamma'], kernel=grid.best_params_['kernel'], probability=True)
model.fit(X_train, y_train)
```

Out[8]:
```
SVC(C=0.1, gamma=0.1, kernel='sigmoid', probability=True)
```

Visualization

I got the prediction probabilities of the trained model using the .predict_proba() function. The plot below shows the distribution of the class probabilities and the default decision boundary.

```
# Get prediction probability for the positive class (hence [:,1])
y_score = model.predict_proba(X_test)[:,1]

# Match the probabilities with the actual class labels
positive_class_distribution = y_score[y_test == 1]
negative_class_distribution = y_score[y_test == 0]

# Plot histogram with decision boundary
plt.figure(figsize=(7,3))
plt.hist(positive_class_distribution, color='mediumaquamarine', alpha=0.6)
plt.hist(negative_class_distribution, color='salmon', alpha=0.6)
plt.axvline(x = 0.5, color = 'k', label = 'axvline - full height')
plt.title("Prediction Histograms at the Default Decision Boundary at 0.5")
plt.xlabel("Prediction Probability")
plt.ylabel("Count")

plt.show()
```

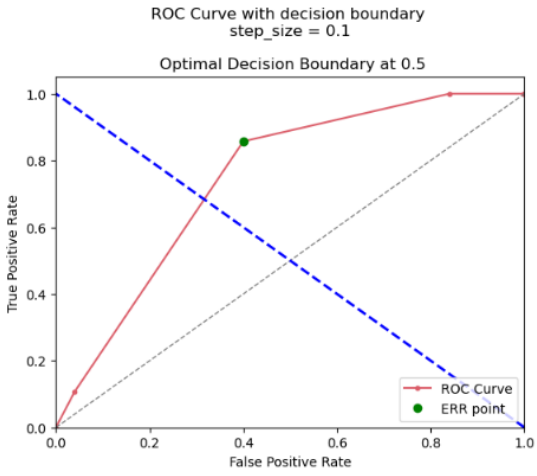


Plotting ROC Curves

In order to plot the ROC Curve (manually), we need to slide the decision boundary across the probability domain (0 to 1). At each step, we will calculate the True Positive Rate and False Positive Rate using the equations below, and plot the generated values against each other to generate the ROC Curve.

$$TPR = \frac{TP}{TP + FN} \qquad FPR = \frac{FP}{FP + TN}$$

```
# List to store values of FPR, TPR, and AUC for different thresholds
fpr_vals = []
tpr_vals = []
thresholds = []

# Step size
step_size = 0.1

# Vary threshold from 0 to 1 with step size of 0.1
for threshold in np.arange(0, 1.1, step_size):
    # Convert predicted probabilities into class labels based on threshold
    y_pred_threshold = np.where(y_score >= threshold, 1, 0)

    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_threshold).ravel()
    tpr = tp / (tp + fn)
    fpr = fp / (fp + tn)

    # Store FPR and TPR values
    fpr_vals.append(fpr)
    tpr_vals.append(tpr)
    thresholds.append(threshold)

# Find the threshold with the lowest ERR
err_index = np.argmin(np.abs((1 - np.array(tpr_vals)) - np.array(fpr_vals)))
optimal_threshold = thresholds[err_index]

# Plot resulting ROC curve
plt.figure()
plt.plot(fpr_vals, tpr_vals, c='#db5c68', marker='o', markersize=3)
plt.plot(fpr_vals[err_index], tpr_vals[err_index],'go')

plt.plot([0, 1], [0, 1], color='gray', lw=1, linestyle='--')
plt.plot([1, 0], [0, 1], color='blue', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve with decision boundary \n step_size = 0.1 \n\n Optimal Decision Boundary at {optimal_threshold}')
plt.legend(["ROC Curve", 'ERR point'], loc='lower right')
plt.show()
```

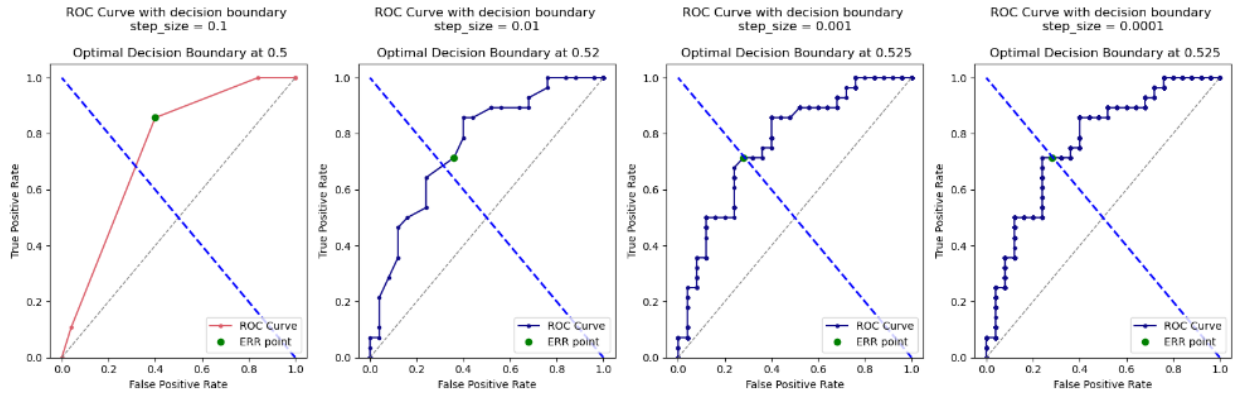


We can then find the equal error rate (ERR) by finding the point in the ROC Curve where:

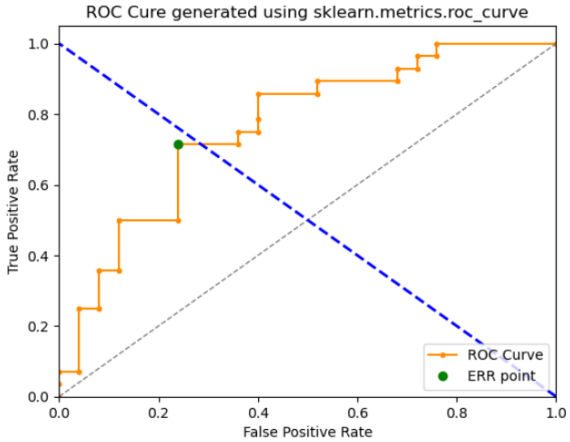
$$TPR = 1 - FPR$$

The threshold value corresponding to this point is the optimal decision boundary that minimizes ERR.

By making the step_size smaller and smaller, we get a more refined and precise ROC curve that better represents the behavior of the model.



We can also generate the ROC curve easily using the sklearn library (sklearn.metrics.roc_curve). We can see that the result generated below is exactly similar with the ROC curve we manually generated using a step size of 0.0001.



Finally, the hyperparameters that define the model also influences the ROC curve. By varying these values, we can a get different ROC Curve.