Advanced Machine Learning

CIS550

Spring ‘24

Lab Homework 6

Submitted by :

Name : Manas Vishal (01971464)

Email : mvishal@umassd.edu

Title : Evaluating a model

Deploying a model

This lab is continuation of previous labs 4 and 5 where we trained a model. In the previous lab, we used the XGBoost library to train a model on the biomedical dataset and deployed it to the dataset but we also found that the model was not that good for some data. In this lab we aim to evaluate the model and its efficiency by different metrics

About the data

This biomedical dataset was built by Dr. Henrique da Mota during a medical residence period in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France. The data has been organized in two different, but related, classification tasks.

The first task consists in classifying patients as belonging to one of three categories:

- *Normal* (100 patients)
- *Disk Hernia* (60 patients)
- *Spondylolisthesis* (150 patients)

For the second task, the categories *Disk Hernia* and *Spondylolisthesis* were merged into a single category that is labeled as *abnormal*. Thus, the second task consists in classifying patients as belonging to one of two categories: *Normal* (100 patients) or *Abnormal* (210 patients).

Each patient is represented in the dataset by six biomechanical attributes that are derived from the shape and orientation of the pelvis and lumbar spine (in this order):

- Pelvic incidence
- Pelvic tilt
- Lumbar lordosis angle
- Sacral slope
- Pelvic radius
- Grade of spondylolisthesis

The following convention is used for the class labels:

- DH (Disk Hernia)
- Spondylolisthesis (SL)
- Normal (NO)
- Abnormal (AB)

For more information about this dataset, see the [Vertebral Column dataset webpage](http://archive.ics.uci.edu/ml/datasets/Vertebral+Column).

Loading the data and deploying the XGBoost model

We follow the same steps as in lab 4 and 5 to load the data. We also train and deploy the model in batch prediction.

```
Predicted class :

0    1
1    1
2    1
3    1
4    1
Name: class, dtype: int64
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Fig. 1 Predicted class compared to the actual class

As we can see the model can be wrong sometimes so in this lab we evaluate the model and classify it with some metrics.

We start by making a confusion matrix.

Step 1

A confusion matrix is a table used in the field of machine learning to evaluate the performance of a classification model. It allows visualization of the performance of an algorithm by comparing predicted classes against actual classes.

We make a confusion matrix with the help of *confusion_matrix* from *sklearn* library.

The values inside the confusion matrix depends on the threshold parameter.

```python
from sklearn.metrics import confusion_matrix

matrix = confusion_matrix(test_labels, target_predicted_binary)
df_confusion = pd.DataFrame(matrix, index=['Normal','Abnormal'],columns=['Normal','Abnormal'])

df_confusion
```

[39] ✓ 0.0s Python

	Normal	Abnormal
Normal	7	3
Abnormal	2	19

Fig. 2 Confusion matrix for threshold 0.5

From the confusion matrix we can infer that the model **correctly** predicted 7 Normal values and 19 Abnormal values, while it **incorrectly** predicted 3 Normal and 2 Abnormal values.

We can also make a heatmap chart from seaborn library.

```python
import seaborn as sns
import matplotlib.pyplot as plt

colormap = sns.color_palette("BrBG", 10)
sns.heatmap(df_confusion, annot=True, cbar=None, cmap=colormap)
plt.title("Confusion Matrix")
plt.tight_layout()
plt.ylabel("True Class")
plt.xlabel("Predicted Class")
plt.show()
```

[40] ✓ 0.2s Python

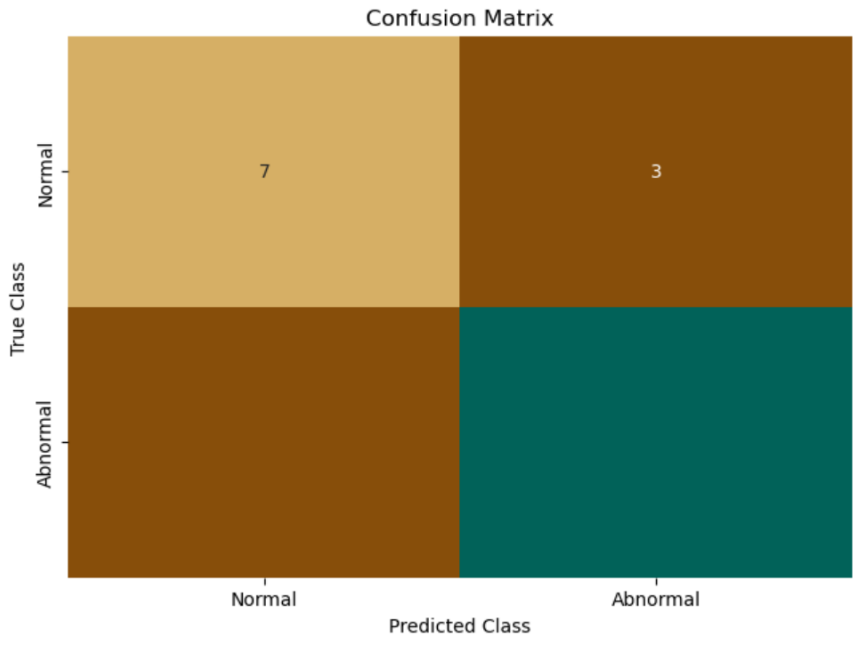

Fig. 3 Heatmap of the confusion matrix

Step 2

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

TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel()

print(f"True Negative (TN) : {TN}")
print(f"False Positive (FP): {FP}")
print(f"False Negative (FN): {FN}")
print(f"True Positive (TP) : {TP}")
```

[41] ✓ 0.0s Python

```
True Negative (TN) : 7
False Positive (FP): 3
False Negative (FN): 2
True Positive (TP) : 19
```

Fig. 4 Confusion matrix values

From these values, we can calculate different statistics like Sensitivity, Specificity, Precision, False Positive Rate, False Negative Rate, False Discovery Rate, Accuracy.

Sensitivity is also known as positivity rate. It is the proportion of actual positive cases that are correctly identified by the classifier.

Sensitivity

Sensitivity is also known as *hit rate*, *recall*, or *true positive rate (TPR)*. It measures the proportion of the actual positives that are correctly identified.

In this example, the sensitivity is *the probablity of detecting an abnormality for patients with an abnormality*.

```
# Sensitivity, hit rate, recall, or true positive rate
Sensitivity  = float(TP)/(TP+FN)*100
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"There is a {Sensitivity}% chance of detecting patients with an abnormality have an abnormality")
```

[14] ✓ 0.0s Python

```
Sensitivity or TPR: 90.47619047619048%
There is a 90.47619047619048% chance of detecting patients with an abnormality have an abnormality
```

Fig. 5 Sensitivity calculation

Question: Is the sensitivity good enough for this scenario?
Answer: Sensitivity is quite high for this classifier so this is good enough however, sensitivity alone can not be used a metric to say quantitatively about the model.

Specificity is defined as the proportion of actual negatives that are correctly identified. It is also known as true negative.

```
# Specificity or true negative rate
Specificity  = float(TN)/(TN+FP)*100
print(f"Specificity or TNR: {Specificity}%")
print(f"There is a {Specificity}% chance of detecting normal patients are normal.")

```

[15] ✓ 0.0s Python

```
Specificity or TNR: 70.0%
There is a 70.0% chance of detecting normal patients are normal.
```

Fig. 6 Specificity calculation

Question: Is this specificity too low, exactly right, or too high? What value would you want to see here, given the scenario?
This specificity is too low. Ideally the specificity should be 100% which means that the model has no false positives. But if the specificity lies in a range say 85 to 95% it will be good enough.

Precision or positive predictive value is the proportion of positive results i.e., ratio of true positives to total positives.

```
# Precision or positive predictive value
Precision = float(TP)/(TP+FP)*100
print(f"Precision: {Precision}%")
print(f"You have an abnormality, and the probablity that is correct is {Precision}%")
```

[16] ✓ 0.0s Python

```
Precision: 86.36363636363636%
You have an abnormality, and the probablity that is correct is 86.36363636363636%
```

Fig. 7 Precision calculation

Negative predictive value is the ratio of negative results to total negative results.

```
# Negative predictive value
NPV = float(TN)/(TN+FN)*100
print(f"Negative Predictive Value: {NPV}%")
print(f"You don't have an abnormality, but there is a {NPV}% chance that is incorrect" )
```

[17] ✓ 0.0s Python

```
Negative Predictive Value: 77.77777777777779%
You don't have an abnormality, but there is a 77.77777777777779% chance that is incorrect
```

Fig. 8 Negative predictive value calculation

From the negative predictive value statistic we can see the model is ill performed in various cases so as a patient I will not be satisfied with these results. I would personally choose 85% as a threshold but scientifically 2 sigma fluctuation or 95% is good model.

False positive rate (FPR) is the probability that a false alarm will be raised, or that a positive result will be given when the true value is negative.

```
[18]
# Fall out or false positive rate
FPR = float(FP)/(FP+TN)*100
print( f"False Positive Rate: {FPR}%")
print( f"There is a {FPR}% chance that this positive result is incorrect.")
```

✓ 0.0s Python

```
False Positive Rate: 30.0%
There is a 30.0% chance that this positive result is incorrect.
```

Fig. 9 False positive rate calculation

False negative rate (FNR) or miss rate is the probability that a true positive will be missed by the test.

```
[19]
# False negative rate
FNR = float(FN)/(TP+FN)*100
print(f"False Negative Rate: {FNR}%")
print(f"There is a {FNR}% chance that this negative result is incorrect.")
```

✓ 0.0s Python

```
False Negative Rate: 9.523809523809524%
There is a 9.523809523809524% chance that this negative result is incorrect.
```

Fig. 10 False negative rate calculation

False discovery rate (FDR) is a statistical measure used in hypothesis testing and multiple comparisons to control the rate of false positives or Type I errors. In the context of binary classification tasks, the false discovery rate

is the proportion of instances predicted as positive by the classifier that are actually negative. It is defined as the ratio of false positive to total positives

```
# False discovery rate
FDR = float(FP)/(TP+FP)*100
print(f"False Discovery Rate: {FDR}%" )
print(f"You have an abnormality, but there is a {FDR}% chance this is incorrect.")
```

[20] ✓ 0.0s Python

```
False Discovery Rate: 13.636363636363635%
You have an abnormality, but there is a 13.636363636363635% chance this is incorrect.
```

Fig. 11 False discovery rate calculation

```
# Overall accuracy
ACC = float(TP+TN)/(TP+FP+FN+TN)*100
print(f"Accuracy: {ACC}%")
```

[21] ✓ 0.0s Python

```
Accuracy: 83.87096774193549%
```

Fig. 12 Accuracy calculation

```
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"Specificity or TNR: {Specificity}%")
print(f"Precision: {Precision}%")
print(f"Negative Predictive Value: {NPV}%")
print( f"False Positive Rate: {FPR}%")
print(f"False Negative Rate: {FNR}%")
print(f"False Discovery Rate: {FDR}%" )
print(f"Accuracy: {ACC}%")
```

[22] ✓ 0.0s Python

```
Sensitivity or TPR: 90.47619047619048%
Specificity or TNR: 70.0%
Precision: 86.36363636363636%
Negative Predictive Value: 77.77777777777779%
False Positive Rate: 30.0%
False Negative Rate: 9.523809523809524%
False Discovery Rate: 13.636363636363635%
Accuracy: 83.87096774193549%
```

Fig. 13 Statistics summary

Challenge Task -1

Threshold =0.25

```
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"Specificity or TNR: {Specificity}%")
print(f"Precision: {Precision}%")
print(f"Negative Predictive Value: {NPV}%")
print( f"False Positive Rate: {FPR}%")
print(f"False Negative Rate: {FNR}%")
print(f"False Discovery Rate: {FDR}%" )
print(f"Accuracy: {ACC}%")
```

[22] ✓ 0.0s Python

```
Sensitivity or TPR: 90.47619047619048%
Specificity or TNR: 70.0%
Precision: 86.36363636363636%
Negative Predictive Value: 77.77777777777779%
False Positive Rate: 30.0%
False Negative Rate: 9.523809523809524%
False Discovery Rate: 13.636363636363635%
Accuracy: 83.87096774193549%
```

Fig. 14 Statistics summary for 0.25 threshold

```
print(f"Sensitivity or TPR: {Sensitivity}%")
print(f"Specificity or TNR: {Specificity}%")
print(f"Precision: {Precision}%")
print(f"Negative Predictive Value: {NPV}%")
print( f"False Positive Rate: {FPR}%")
print(f"False Negative Rate: {FNR}%")
print(f"False Discovery Rate: {FDR}%" )
print(f"Accuracy: {ACC}%")
```

[36] ✓ 0.0s Python

```
Sensitivity or TPR: 85.71428571428571%
Specificity or TNR: 70.0%
Precision: 85.71428571428571%
Negative Predictive Value: 70.0%
False Positive Rate: 30.0%
False Negative Rate: 14.285714285714285%
False Discovery Rate: 14.285714285714285%
Accuracy: 80.64516129032258%
```

Fig. 15 Statistics summary for 0.75 threshold

The threshold does make a difference as from previous lab we saw that there was a cutoff for probability hence the confusion matrix changes with change in the threshold. Increasing the threshold lowered the accuracy.

Step 3

We calculate the AUC-ROC. AUC stands for Area Under the ROC Curve. ROC stands for Receiver Operating Characteristic. AUC and ROC curve are performance evaluation metrics used in binary classification tasks to assess the predictive power of a classifier.

ROC is a probability curve. The AUC tells us how well the model distinguishes between classes.

```
[23]  ✓ 0.0s                                                    Python

1  test_labels = test.iloc[:,0];
2  print("Validation AUC", roc_auc_score(test_labels, target_predicted) )

...  Validation AUC 0.8857142857142857
```

Fig. 16 AUC calculation

```
1  ✓ 0.2s                                                       Python

import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming test_labels and target_predicted are defined

fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r')
ax2.set_ylabel('Threshold', color='r')
ax2.set_xlim([fpr[0], fpr[-1]])

plt.show()

```

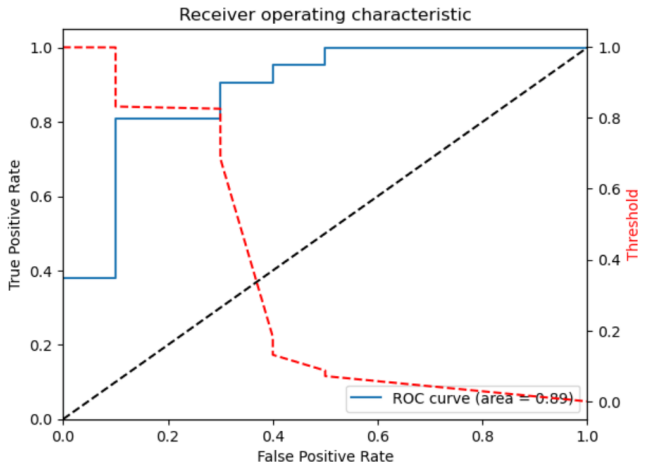

Fig. 17 ROC curve

Challenge Task 2

If we change the code to use binary values, the AUC decreases and the model it is less useful than the previous case.

```python
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming test_labels and target_predicted are defined

fpr, tpr, thresholds = roc_curve(test_labels, target_predicted_binary)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r')
ax2.set_ylabel('Threshold', color='r')
ax2.set_xlim([fpr[0], fpr[-1]])

plt.show()
```

40] ✓ 0.2s Python

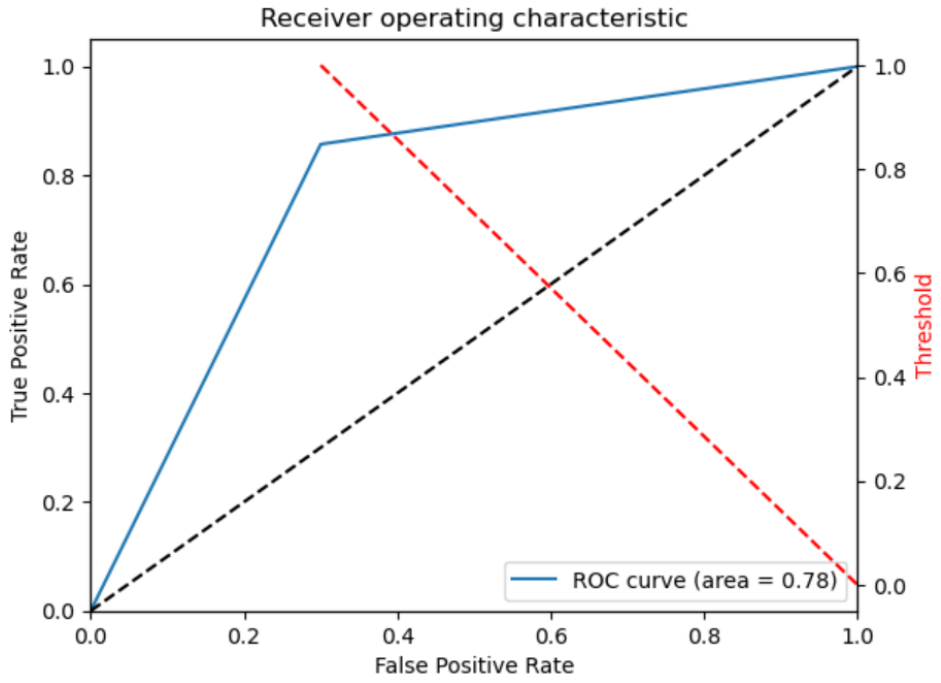

Fig. 18 ROC curve for binary values

Conclusion

In conclusion, the model I evaluated in this Advanced Machine Learning Lab homework provided me with valuable insight into its performance using

various statistical metrics and visualization techniques. During the analysis, I focused on evaluating the predictive power of the model and its ability to distinguish between different classes of biomedical datasets.

I used metrics such as negative predictive value, false positive rate, false negative rate and false discovery rate, which helped to understand the accuracy and reliability of the model. The choice of thresholds played a crucial role in shaping the results of the model, as different thresholds led to variation in the confusion matrix and overall accuracy.

In addition, calculating the area under the ROC curve (AUC-ROC) gave me comprehensive assessment of the predictive ability of the model in binary classification tasks. The ROC curve visually represented the trade-off between the true positive rate and the false positive rate, giving me a clear idea of the model's performance at different thresholds.

I also noticed the effect of using binary values on the AUC, and observed a decrease in the utility of the model compared to the initial estimate in such cases. This finding underscored the importance of choosing appropriate data representations and thresholds to optimize model performance.

Overall, the model evaluation performed in Manas Vishal's lab homework highlighted the importance of different evaluation metrics and visualization tools to assess relevance. the effectiveness of machine learning models in real-world applications. Insights from this evaluation process can guide future model improvements and decision-making processes in advanced machine learning and biomedical data analysis.