Name : MIN SOE HTUT

ID : 1631938

Part A

In [111]:
```
import numpy as np
import pandas as pd
import sklearn.metrics as skmetric
url = 'https://raw.githubusercontent.com/bpfa/data_for_compx310_2023/main/wisc
onsin_breast_cancer.csv'
df = pd.read_csv(url)
df
```

Out[111]:

	id	thickness	size	shape	adhesion	single	nuclei	chromatin	nucleoli	mitosis	clas
0	1000025	5	1	1	1	2	1.0	3	1	1	
1	1002945	5	4	4	5	7	10.0	3	2	1	
2	1015425	3	1	1	1	2	2.0	3	1	1	
3	1016277	6	8	8	1	3	4.0	3	7	1	
4	1017023	4	1	1	3	2	1.0	3	1	1	
...	...	...	...	...	...	...	...	...	...	...	
694	776715	3	1	1	1	3	2.0	1	1	1	
695	841769	2	1	1	1	2	1.0	1	1	1	
696	888820	5	10	10	3	7	3.0	8	10	2	
697	897471	4	8	6	4	3	4.0	10	6	1	
698	897471	4	8	8	5	4	5.0	10	4	1	

699 rows × 11 columns

In [112]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   id         699 non-null    int64
 1   thickness  699 non-null    int64
 2   size       699 non-null    int64
 3   shape      699 non-null    int64
 4   adhesion   699 non-null    int64
 5   single     699 non-null    int64
 6   nuclei     683 non-null    float64
 7   chromatin  699 non-null    int64
 8   nucleoli   699 non-null    int64
 9   mitosis    699 non-null    int64
 10  class      699 non-null    int64
dtypes: float64(1), int64(10)
memory usage: 60.2 KB
```

Resolving the missing values, Selecting all features except 'ID' and 'class' as X and selecting "class" as y

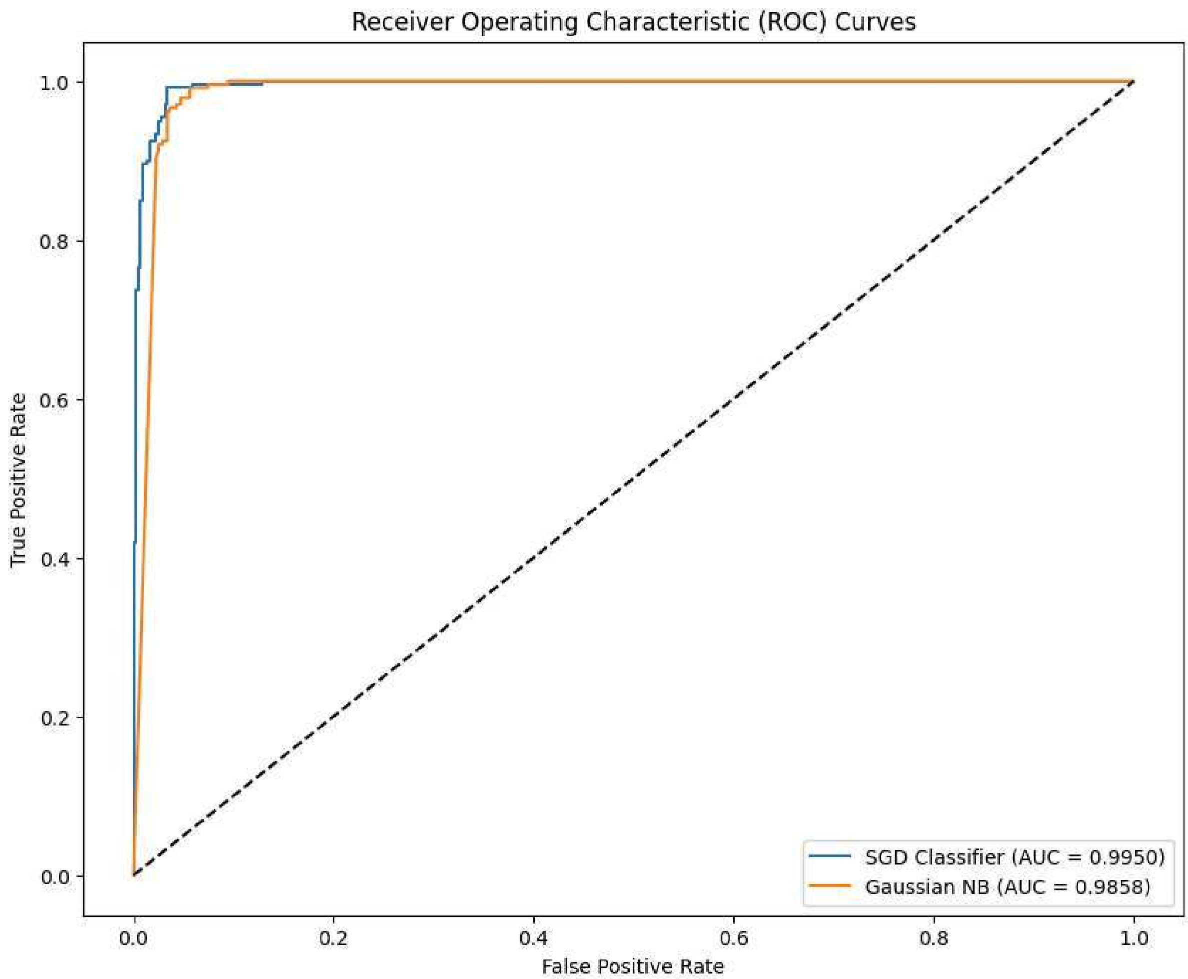
In [113]:
```
ID = 1631938
# Drop rows with missing values
df = df.dropna()
# Separate features and target variable
X = df.iloc[:, 1:-1]
y = df.iloc[:, -1]
```

Useing 5-fold cross-validation to generate predictions from the following classifiers: Use the top SGDClassifier,GaussianNB.

In [114]:
```
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_predict, cross_val_score

# Initialize classifiers
sgd_classifier = SGDClassifier(random_state= ID)
gaussian_nb_classifier = GaussianNB()

# Generate predicted scores
y_sgd_score = cross_val_predict(sgd_classifier, X, y, cv=5, method='decision_f
unction')
y_gaussian_nb_score = cross_val_predict(gaussian_nb_classifier, X, y, cv=5, me
thod='predict_proba')[:, 1]
```

Plotting ROC Curves

In [115]:
```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt


# Part B: Compute ROC curve
sgd_fpr, sgd_tpr, _ = roc_curve(y, y_sgd_score)
gaussian_nb_fpr, gaussian_nb_tpr, _ = roc_curve(y, y_gaussian_nb_score)

# Part C: Compute AUC values
sgd_auc = roc_auc_score(y, y_sgd_score)
gaussian_nb_auc = roc_auc_score(y, y_gaussian_nb_score)

# Plot ROC curves
plt.figure(figsize=(10, 8))
plt.plot(sgd_fpr, sgd_tpr, label=f'SGD Classifier (AUC = {sgd_auc:.4f})')
plt.plot(gaussian_nb_fpr, gaussian_nb_tpr, label=f'Gaussian NB (AUC = {gaussia
n_nb_auc:.4f})')

# Plot diagonal line
plt.plot([0, 1], [0, 1], 'k--')

# Add labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend()
plt.show()
```



Finding the cross validation accuracy of the classifiers using cross_val_score

In [116]:
```
from sklearn.metrics import classification_report, confusion_matrix
# Find cross-validation accuracy
sgd_accuracy = cross_val_score(sgd_classifier, X, y, cv=5).mean()
gaussian_nb_accuracy = cross_val_score(gaussian_nb_classifier, X, y, cv=5).mea
n()

print(f'SGD Classifier Cross-validation Accuracy: {sgd_accuracy:.2f}')
print(f'Gaussian NB Cross-validation Accuracy: {gaussian_nb_accuracy:.2f}')

# Classification report and confusion matrix
sgd_predictions = cross_val_predict(sgd_classifier, X, y, cv=5,
method='predict')
gaussian_nb_predictions = cross_val_predict(gaussian_nb_classifier, X,
y, cv=5, method='predict')

# Classification report and confusion matrix
print('SGD Classifier Classification Report:')
print(classification_report(y, sgd_predictions))
print('Gaussian NB Classification Report:')
print(classification_report(y, gaussian_nb_predictions))

print('SGD Classifier Confusion Matrix:')
print(confusion_matrix(y, sgd_predictions))
print('Gaussian NB Confusion Matrix:')
print(confusion_matrix(y, gaussian_nb_predictions))
```

```
SGD Classifier Cross-validation Accuracy: 0.96
Gaussian NB Cross-validation Accuracy: 0.96
SGD Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       444
           1       0.95      0.95      0.95       239

    accuracy                           0.96       683
   macro avg       0.96      0.96      0.96       683
weighted avg       0.96      0.96      0.96       683

Gaussian NB Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.97       444
           1       0.92      0.97      0.94       239

    accuracy                           0.96       683
   macro avg       0.95      0.96      0.96       683
weighted avg       0.96      0.96      0.96       683

SGD Classifier Confusion Matrix:
[[431  13]
 [ 11 228]]
Gaussian NB Confusion Matrix:
[[423  21]
 [  7 232]]
```

Discussion

I prefer the GaussianNB classifier over the SGDClassifier because it has a higher true positive rate, as per the plot by the ROC curve, and has a slightly higher AUC value than SGD Classifier.

The classifier with the best AUC value is not always the most accurate.The GaussianNB classifier has a better AUC value, suggesting it has a superior ability to distinguish between classes across various thresholds.Gaussian NB classifier has the higher accuracy rate at 96% when compared to SGD classifier.

Precision measures the quality of positive predictions by indicating the proportion of true positives among all positive predictions reflecting the classifier's ability to avoid false positives. Recall assesses the classifier's ability to identify all relevant positive cases by showing the proportion of true positives among all actual positives.High precision ensures accurate positive predictions, while high recall ensures that most positive cases are detected.

Part B

Load the New Datasets

In [117]:
```
import pandas as pd

# Load the new datasets
minnesota_df = pd.read_csv('https://raw.githubusercontent.com/nlim-uow/my_note
s/main/test_dataset_minnesota.csv')
melbourne_df = pd.read_csv('https://raw.githubusercontent.com/nlim-uow/my_note
s/main/test_dataset_melbourne.csv')

# Separate features and target variable for both datasets
X_minnesota = minnesota_df.iloc[:, 1:-1]
y_minnesota = minnesota_df.iloc[:, -1]
X_melbourne = melbourne_df.iloc[:, 1:-1]
y_melbourne = melbourne_df.iloc[:, -1]
```

Fit the GaussianNB Model to the Training Data

In [118]:
```
from sklearn.naive_bayes import GaussianNB

# Fit the GaussianNB model to the original training data (from Part A)
gaussian_nb_classifier = GaussianNB()
gaussian_nb_classifier.fit(X, y)
```

Out[118]:
```
▾ GaussianNB
GaussianNB()
```

Generate Predictions and Scores for the Minnesota Dataset

In [119]:
```
# Generate predictions and scores for Minnesota dataset
y_minnesota_pred = gaussian_nb_classifier.predict(X_minnesota)
y_minnesota_score = gaussian_nb_classifier.predict_proba(X_minnesota)[:, 1]
```

Generate Predictions and Scores for the Melbourne Dataset

In [120]:
```
# Generate predictions and scores for Melbourne dataset
y_melbourne_pred = gaussian_nb_classifier.predict(X_melbourne)
y_melbourne_score = gaussian_nb_classifier.predict_proba(X_melbourne)[:, 1]
```

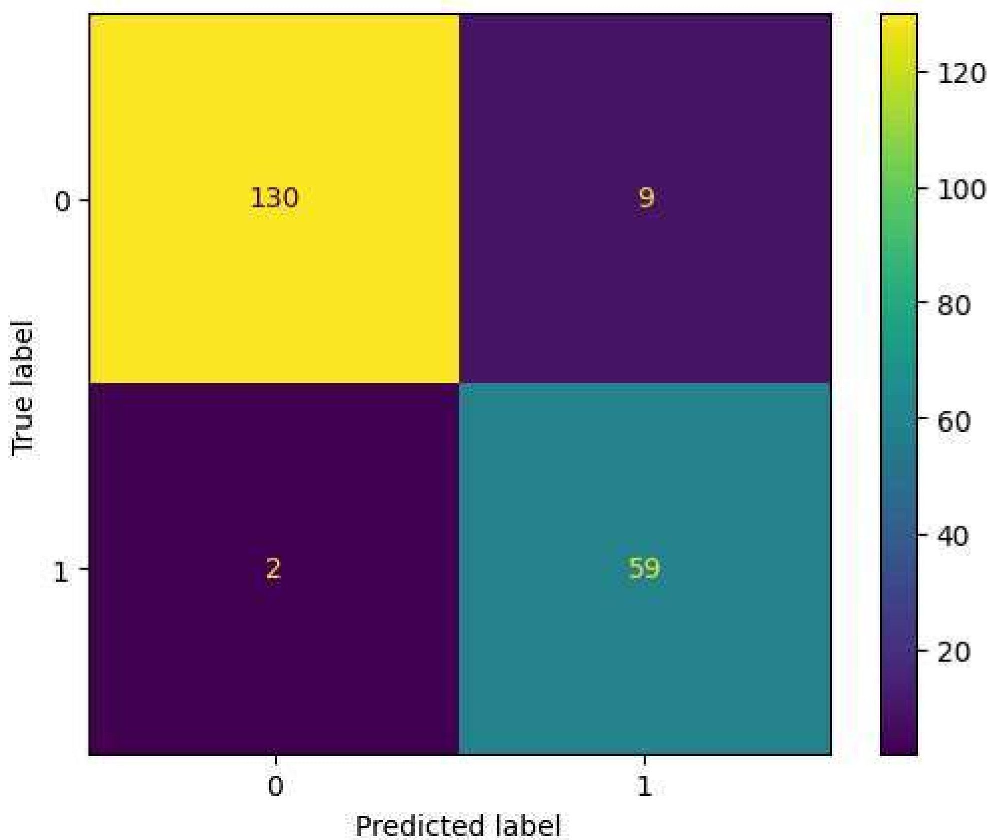
Define a Function to Evaluate Model Performance

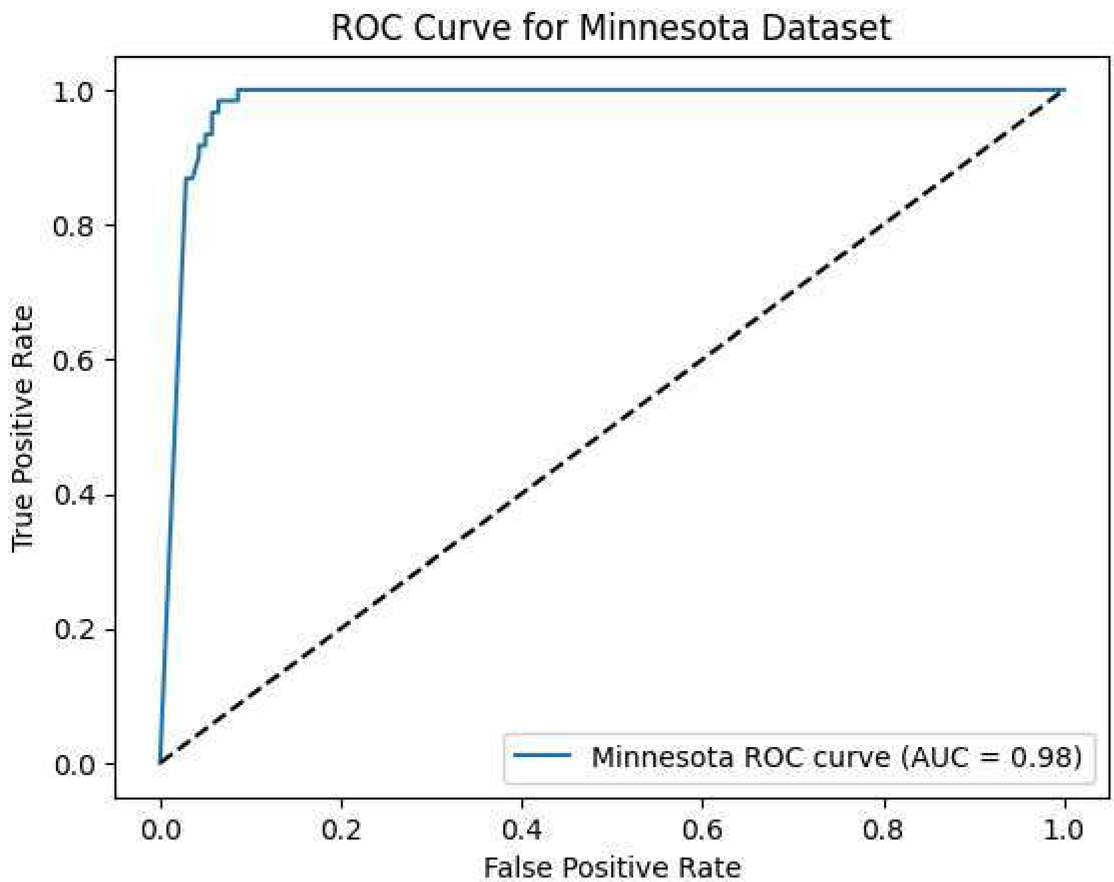
In [121]:
```
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, c
onfusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Function to plot ROC curve and print classification report
def evaluate_model_performance(y_true, y_pred, y_score, dataset_name):
    fpr, tpr, _ = roc_curve(y_true, y_score)
    auc = roc_auc_score(y_true, y_score)
    print(f'Classification Report for {dataset_name} dataset:')
    print(classification_report(y_true, y_pred))
    print(f'Confusion Matrix for {dataset_name} dataset:')
    cm = confusion_matrix(y_true, y_pred)
    ConfusionMatrixDisplay(cm).plot()
    plt.show()
    plt.plot(fpr, tpr, label=f'{dataset_name} ROC curve (AUC = {auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {dataset_name} Dataset')
    plt.legend()
    plt.show()
```

Evaluate Performance on the Minnesota Dataset

In [122]:
```
# Evaluate performance on Minnesota dataset
evaluate_model_performance(y_minnesota, y_minnesota_pred, y_minnesota_score,
'Minnesota')
```

```
Classification Report for Minnesota dataset:
              precision    recall  f1-score   support

           0       0.98      0.94      0.96       139
           1       0.87      0.97      0.91        61

    accuracy                           0.94       200
   macro avg       0.93      0.95      0.94       200
weighted avg       0.95      0.94      0.95       200

Confusion Matrix for Minnesota dataset:
```

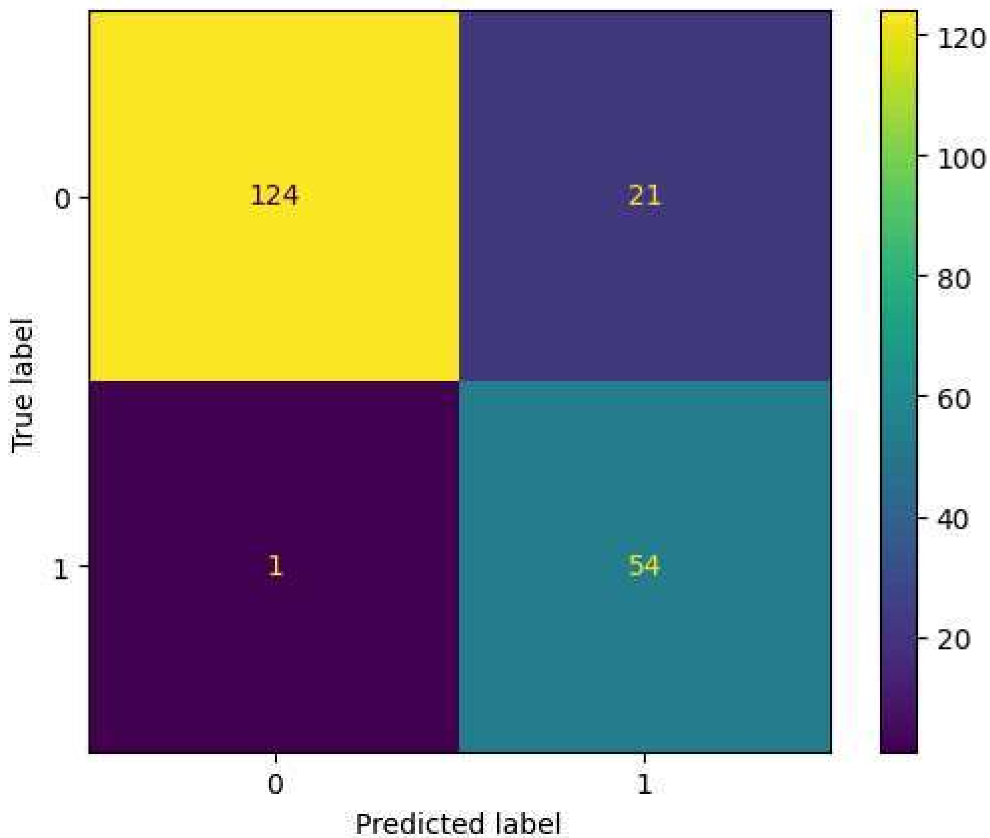


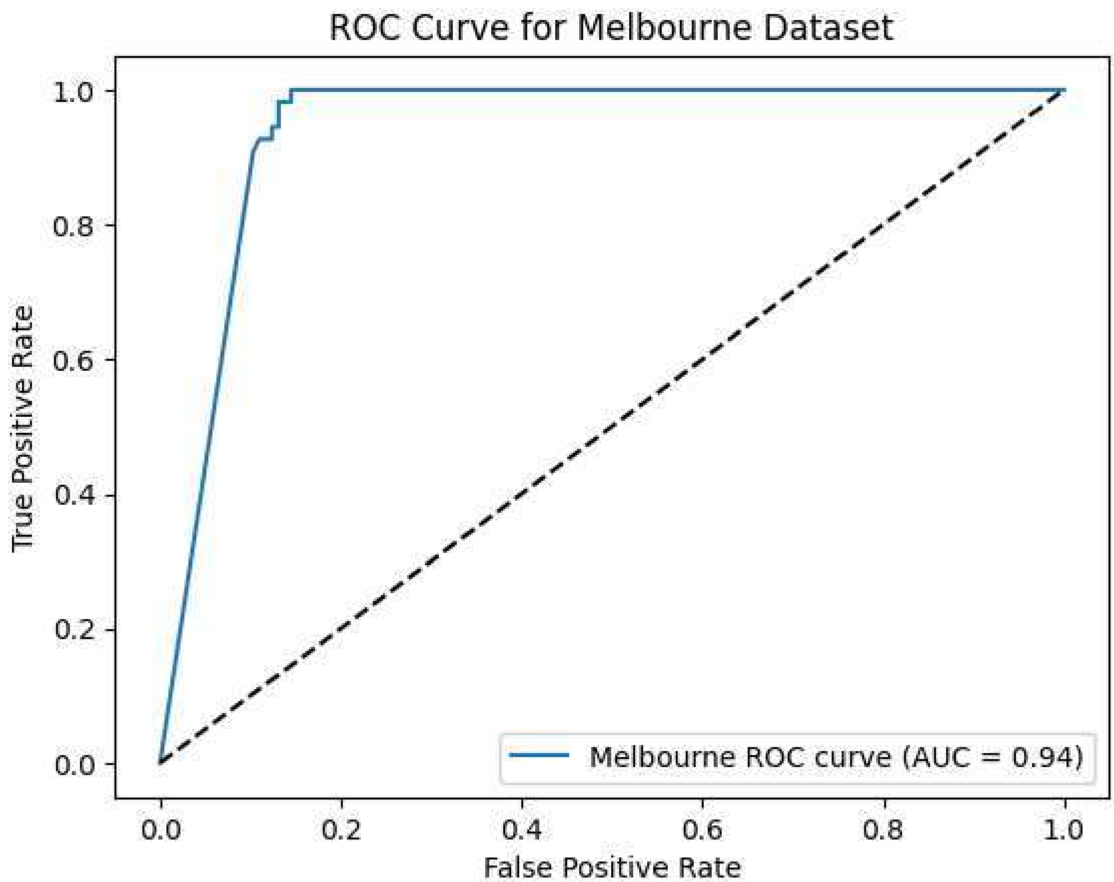


Evaluate Performance on the Melbourne Dataset

In [123]:
```
# Evaluate performance on Melbourne dataset
evaluate_model_performance(y_melbourne, y_melbourne_pred, y_melbourne_score,
'Melbourne')
```

```
Classification Report for Melbourne dataset:
              precision    recall  f1-score   support

           0       0.99      0.86      0.92       145
           1       0.72      0.98      0.83        55

    accuracy                           0.89       200
   macro avg       0.86      0.92      0.87       200
weighted avg       0.92      0.89      0.89       200

Confusion Matrix for Melbourne dataset:
```





Discussion

The differing results likely arise from variations in patient demographics and diagnostic practices between the Minnesota and Melbourne hospitals. Different hospitals might have unique methods of data collection and varying patient populations, leading to differences in the datasets and consequently the model performance .

To address these discrepancies, we can apply techniques such as feature engineering and selection to ensure that the most relevant features are used, and the irrelevant ones are removed. This can help the model generalize better across different datasets. Regularization techniques, such as Ridge or Lasso, can also be employed to prevent overfitting and enhance the model's robustness .

If collecting more data is not an option, we can use domain adaptation techniques, such as transfer learning, where the model is fine-tuned on a small portion of the new dataset. Additionally, employing stronger regularization, hyperparameter tuning, and using simpler models with fewer features can improve the model's generalization to new data .

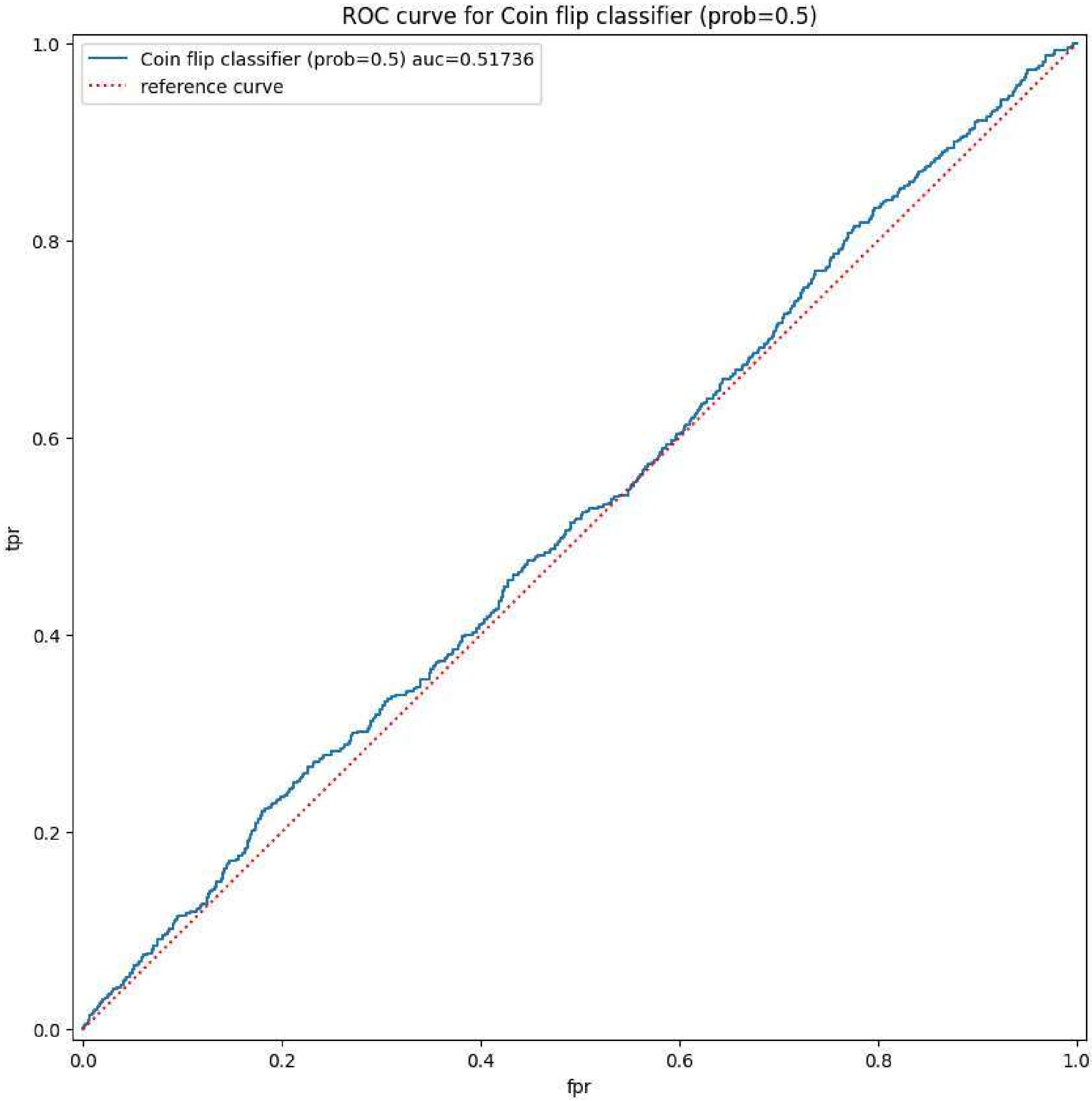
In [124]:
```
import numpy as np
import pandas as pd
import sklearn.metrics as skmetric
import seaborn as sns
# This function rotates the first two features by `angle` degrees, what it doe
s in essence is
# to make the first two features relevant to the classification and while leav
ing the rest of the
# features irrelevant
def rotate_data(X,angle):
    angle=angle/180*np.pi # Convert angles to radians
    transform_array=np.eye(X.shape[1])
    transform_array[0,0]=np.cos(angle)
    transform_array[0,1]=np.sin(angle)
    transform_array[1,0]=-np.sin(angle)
    transform_array[1,1]=np.cos(angle)
    X=np.matmul(X,transform_array)
    return X
# Create some data, this is a "simple" classification problem. All this is, is
a dataframe with
# 10000 observations and 30 features. Only the first feature is relevant for t
he classification
# the data is imbalanced, with `~thr` examples in the negative class, and `~1-
thr` examples in the
# positive class. We intetionally make this an imbalanced dataset where y=1 is
rare (1 in 20)
angle=30
no_dummy_feature=8
thr=0.95
X=np.random.rand(10000,no_dummy_feature+2)
y=X[:,0]>thr
X= rotate_data(X,angle) #by rotating the data, we make it such that the first
2 features are relevant
columns=['relevant_feature_1','relevant_feature_2']
for i in range(no_dummy_feature):
    columns.append('dummy_feature_'+str(i))
columns.append('class')
df = pd.DataFrame(np.concatenate([X,y.reshape(-1,1)],axis=1),columns=columns)
df.info()
# plot the data and visualize it, observe that only the first 2 features are u
seful for the classification
# and for the other features, it is just noise
sns.pairplot(data=df,hue='class')

# Helper function because if you have to call something 50 times, you might as
well write a function
def report_performance(y,y_pred,y_score,label=''):
    fpr,tpr,_=skmetric.roc_curve(y,y_score)
    roc_auc=skmetric.roc_auc_score(y,y_score)
    fig=plt.figure(figsize=(10,10))
    plt.plot(fpr,tpr, label=f'{label} auc={roc_auc:0.5}')
    plt.plot([0,1],[0,1], 'r:',label='reference curve')
    plt.xlabel('fpr')
    plt.ylabel('tpr')
    plt.title(f'ROC curve for {label}')
    plt.xlim([-0.01,1.01])
```

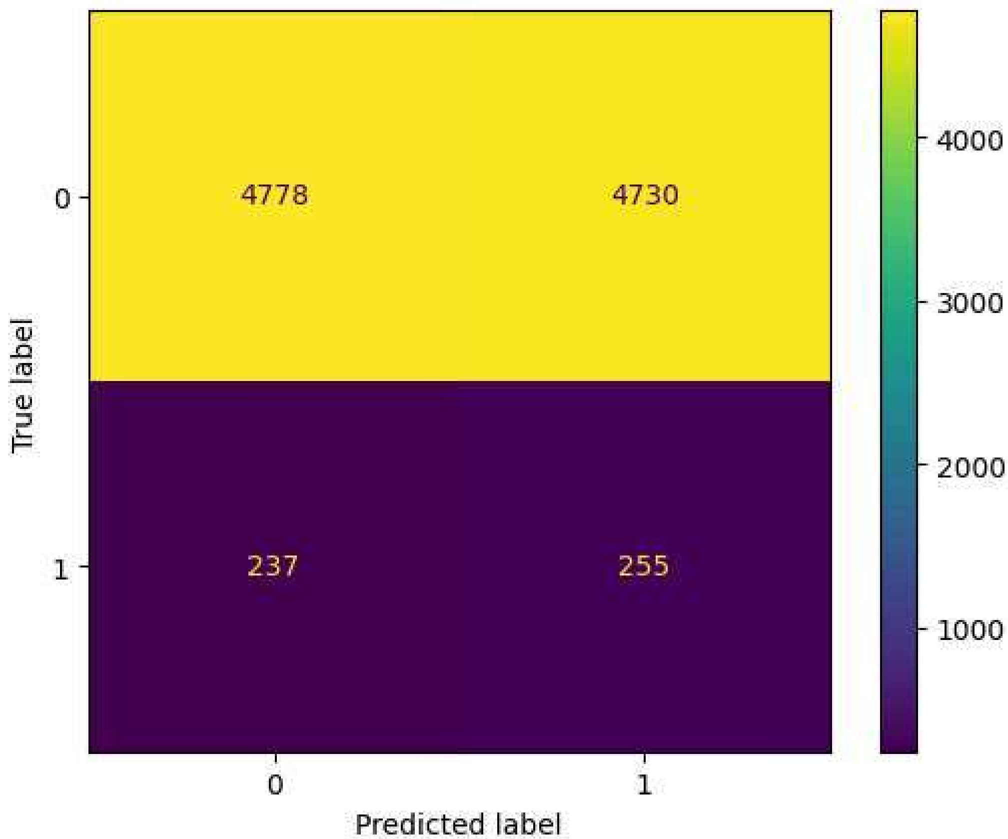
```
    plt.ylim([-0.01,1.01])
    plt.legend()
    plt.show()
    cm = skmetric.confusion_matrix(y, y_pred, labels=[0,1])
    disp = skmetric.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels
=[0,1])
    disp.plot()
    plt.show()
    print(skmetric.classification_report(y,y_pred))

# Let's define some dumb classifier, This is a random classifier that flips an
unfair coin, and votes
# according to the result of the unfair coin, prob sets the probability that t
he classifier would vote
# for class '1'. The scores reported are randomly generated according to the p
redicted class

def coinflip_classifier(X,prob=0.5):
    noQueries=X.shape[0]
    y_score=np.random.uniform(0,1,X.shape[0])
    y_pred = y_score>(1-prob)
    return y_pred,y_score

# This is another dumb classifier, it just votes according to the majority cla
ss, if there are more
# class=0 in the labels, it will always vote 0, otherwise it will always vote
1. The score reported
# is meaningless (it's always a large number). y_score bit looks a bit funky m
ainly to work-around
# some issues roc_curve will have when there is only a single score (it will o
nly be a single point)

def majority_class_classifier(X,y):
    majClass=np.round(sum(y)/X.shape[0])
    y_pred=np.zeros(X.shape[0],dtype=int)+majClass
    y_score=np.ones(X.shape[0])*majClass
    y_score=majClass+0.000001*(1-2*y)
    return y_pred,y_score
# This is what the ideal classifier should look like, it returns y_pred accord
ing to the relevant feature
# (column 0), the score of the prediction is the distance of the point to the
decision boundary;
# This ideal classifier may use the "correct" concept, but may use the wrong t
hreshold.
def ideal_classifier(X,thr=0.5,angle=30):
    angle=angle/180*np.pi
    y_score =X[:,0]*np.cos(angle)+X[:,1]*np.sin(angle)
    y_pred= y_score>thr
    return y_pred,y_score
```

Output hidden; open in https://colab.research.google.com to view.

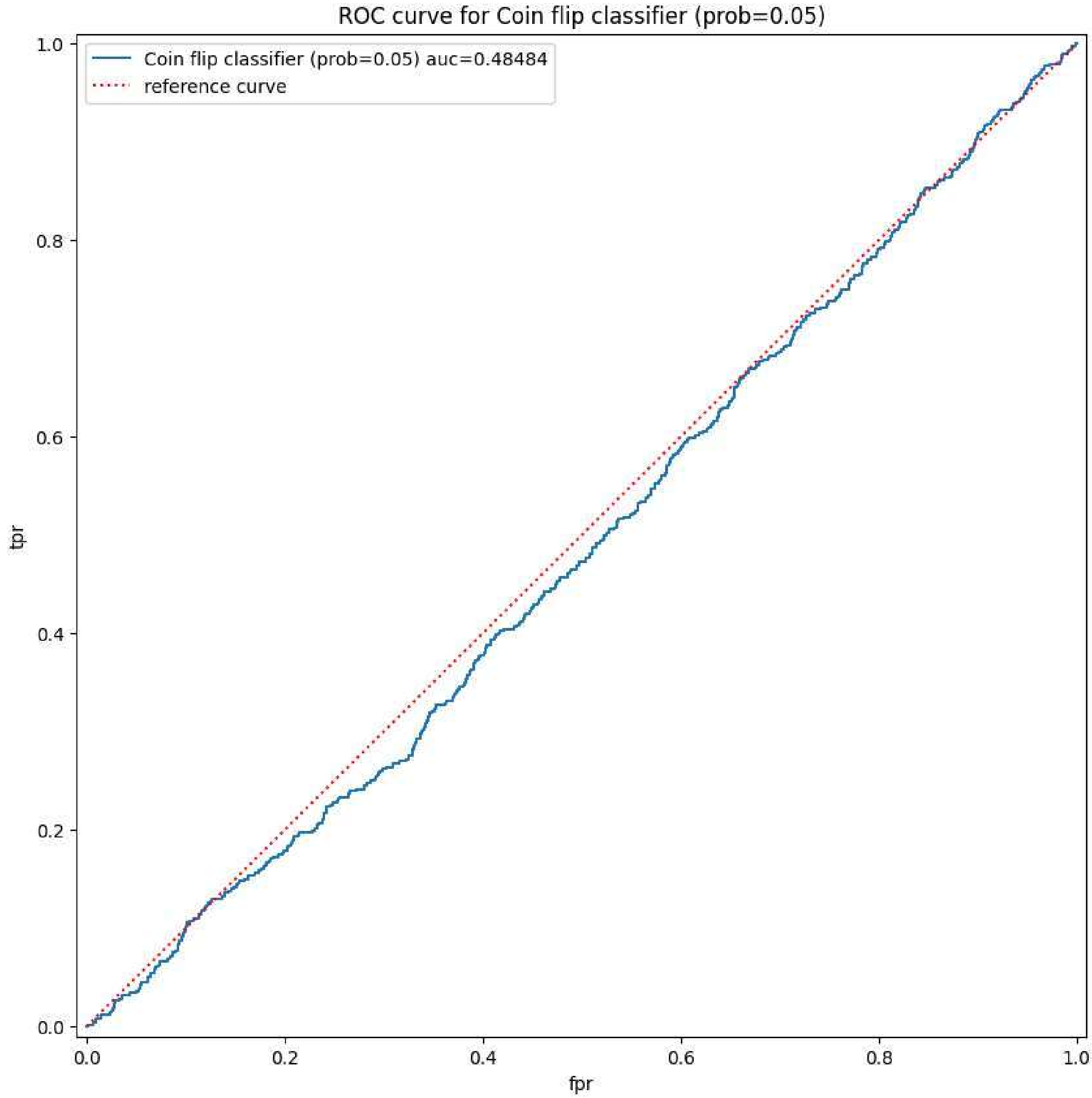
Part C (a)

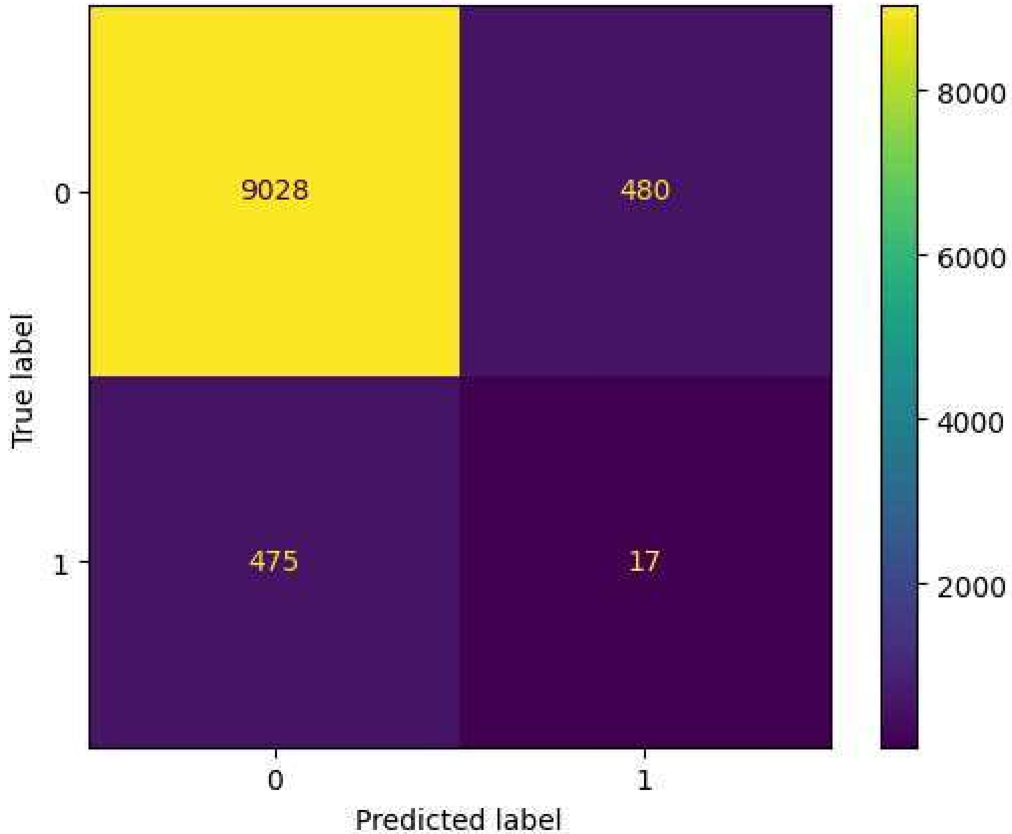
In [125]:
```
y_pred,y_score=coinflip_classifier(X,prob=0.5)
report_performance(y,y_pred,y_score,label=f'Coin flip classifier (prob=0.5)')
y_pred,y_score=coinflip_classifier(X,prob=1-thr)
report_performance(y,y_pred,y_score,label=f'Coin flip classifier (prob={1-thr:0.4})')
```


ROC curve for Coin flip classifier (prob=0.5)
Coin flip classifier (prob=0.5) auc=0.51736
reference curve
tpr
fpr
0.0
0.2
0.4
0.6
0.8
1.0




```
              precision    recall  f1-score   support

       False       0.95      0.50      0.66      9508
        True       0.05      0.52      0.09       492

    accuracy                           0.50     10000
   macro avg       0.50      0.51      0.38     10000
weighted avg       0.91      0.50      0.63     10000
```


ROC curve for Coin flip classifier (prob=0.05)
Coin flip classifier (prob=0.05) auc=0.48484
reference curve
tpr
fpr
0.0
0.2
0.4
0.6
0.8
1.0




```
              precision    recall  f1-score   support

       False       0.95      0.95      0.95      9508
        True       0.03      0.03      0.03       492

    accuracy                           0.90     10000
   macro avg       0.49      0.49      0.49     10000
weighted avg       0.90      0.90      0.90     10000
```

Discussion for Part C (a)

Coin Flip Classifier (prob=0.5)

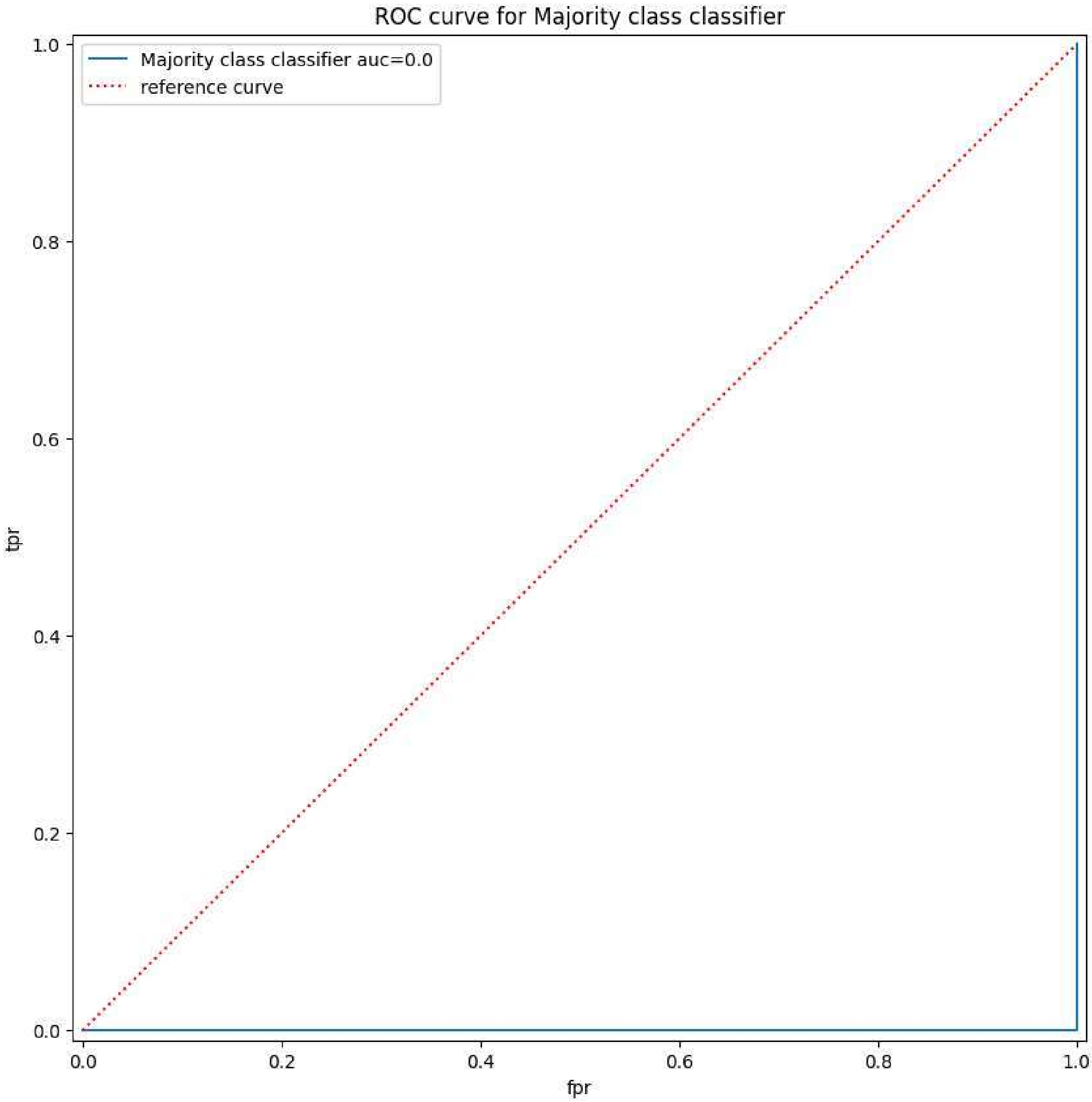
The low precision and recall for the positive class, along with the ROC curve and AUC value, confirm that the coin flip classifier (prob=0.5) performs no better than random guessing.

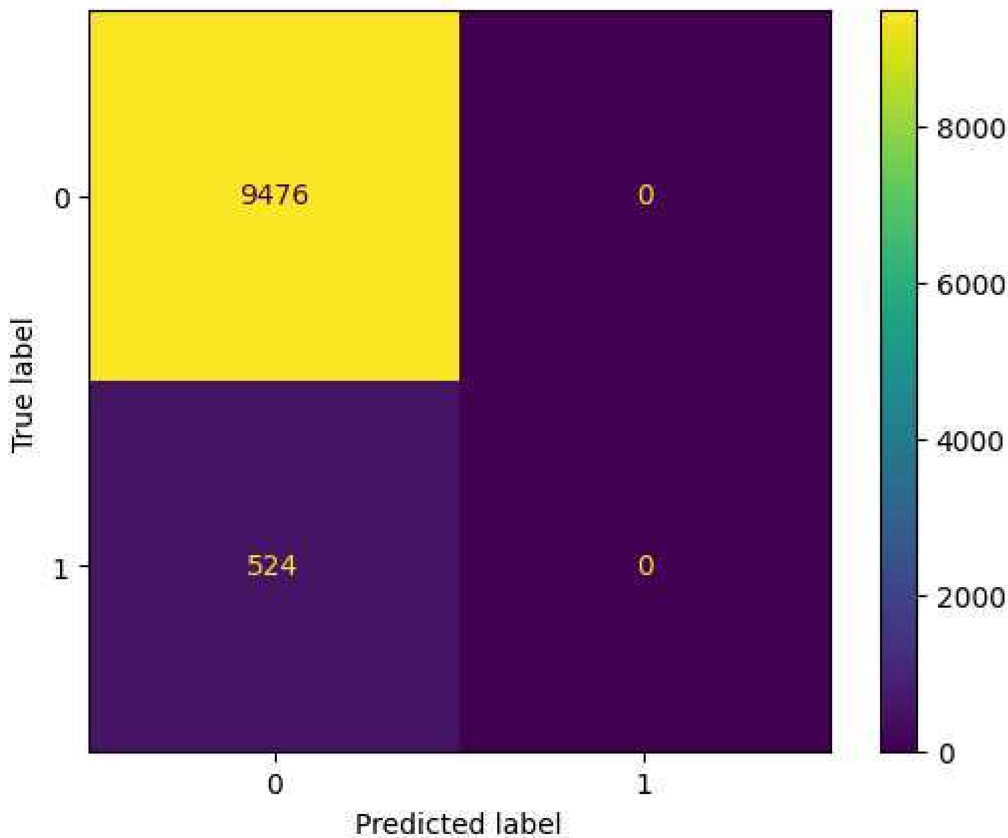
Coin Flip Classifier (prob=1-thr)

The high overall accuracy (90%) is misleading due to the imbalanced dataset. The ROC curve with an AUC of approximately 0.47 and the classification report confirm that the model performs no better than random guessing for positive cases. The low precision and recall highlight the model's inability to identify positive instances effectively.

Part C (b)

In [110]:
```
y_pred,y_score=majority_class_classifier(X,y)
report_performance(y,y_pred,y_score,label=f'Majority class classifier')
```


ROC curve for Majority class classifier
Majority class classifier auc=0.0
reference curve
tpr
fpr
0.0
0.2
0.4
0.6
0.8
1.0




```
              precision    recall  f1-score   support

       False       0.95      1.00      0.97      9476
        True       0.00      0.00      0.00       524

    accuracy                           0.95     10000
   macro avg       0.47      0.50      0.49     10000
weighted avg       0.90      0.95      0.92     10000
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:14
71: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:14
71: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:14
71: UndefinedMetricWarning: Precision and F-score are ill-defined and being s
et to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

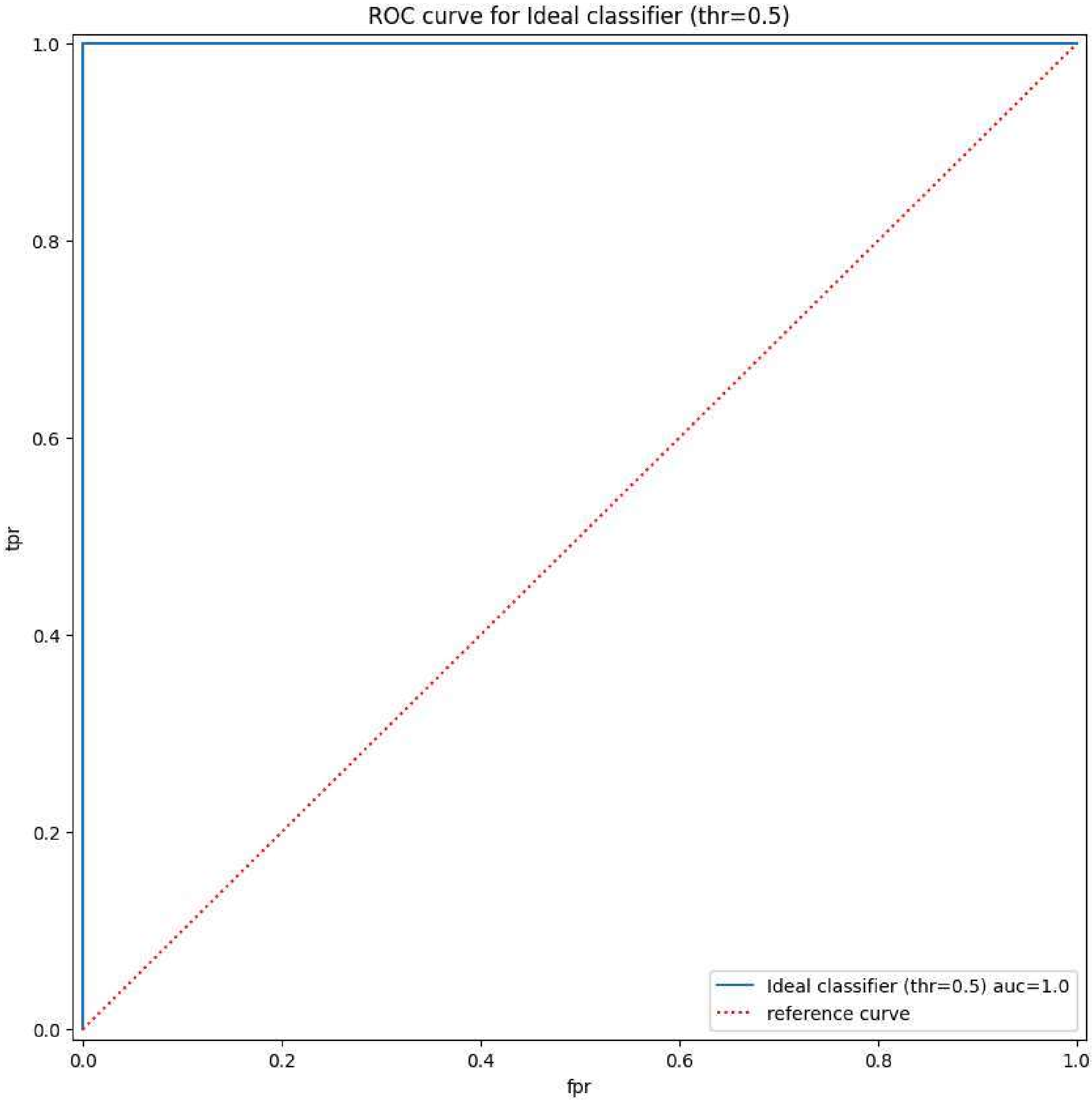
Discussion For Part C (b)

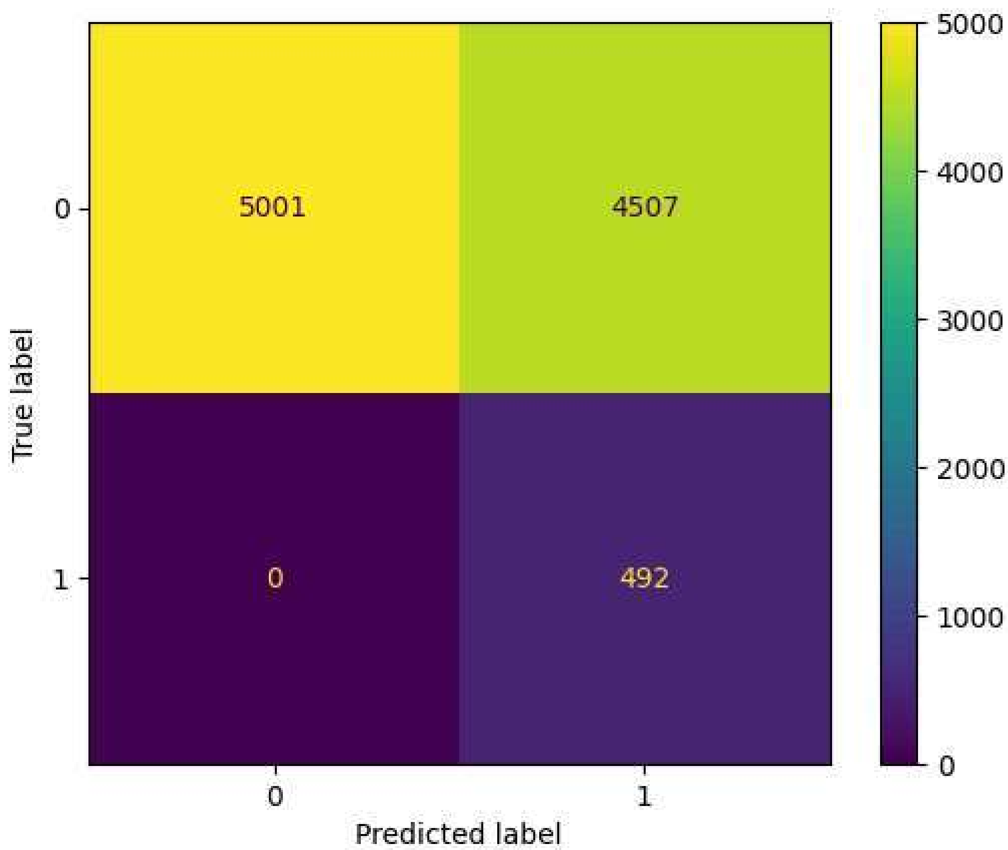
Majority Class Classifier

The high overall accuracy (95%) is misleading due to the imbalanced dataset. The ROC curve with an AUC of 0.0 and the classification report both confirm that the model cannot identify positive cases at all, highlighting the limitation of relying solely on overall accuracy in imbalanced datasets.

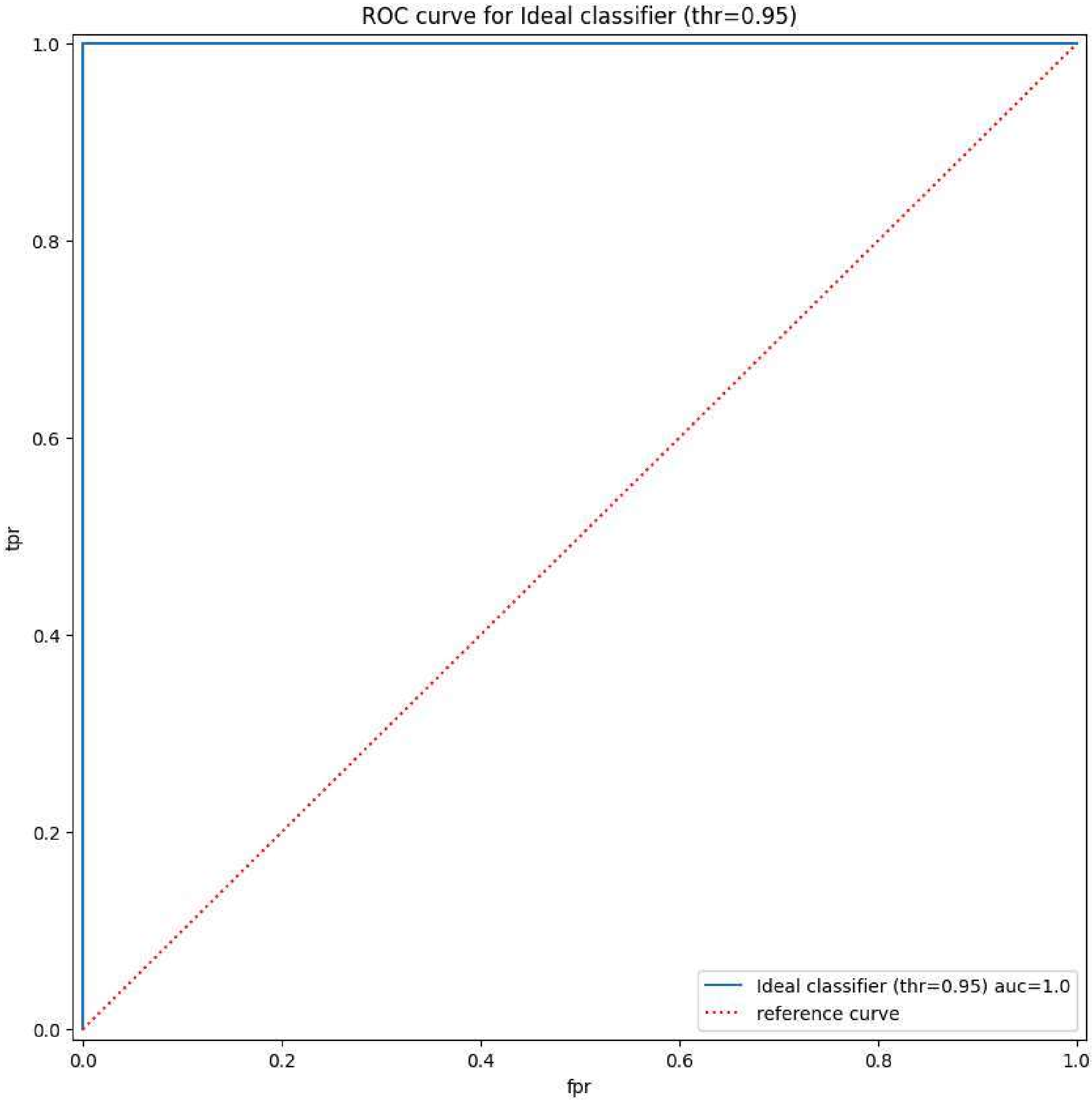
Part C (c)

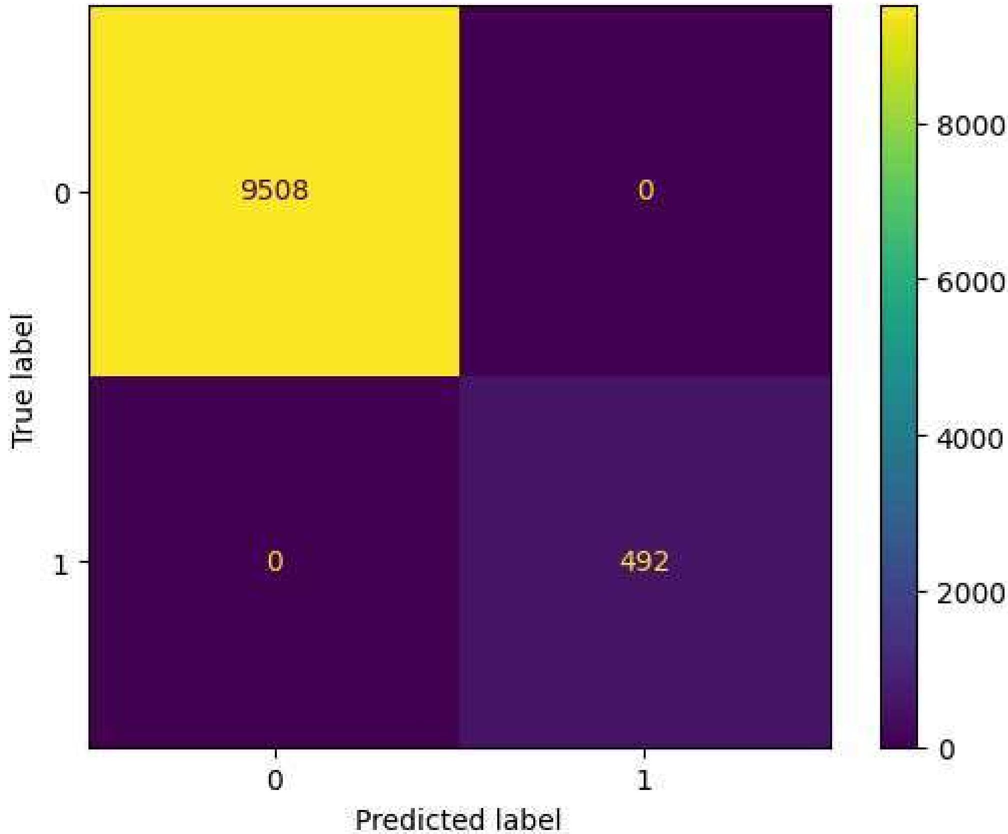
In [126]:
```
y_pred,y_score=ideal_classifier(X,thr=0.5,angle=angle)
report_performance(y,y_pred,y_score,label=f'Ideal classifier (thr=0.5)')
y_pred,y_score=ideal_classifier(X,thr=thr,angle=angle)
report_performance(y,y_pred,y_score,label=f'Ideal classifier (thr={thr:0.3})')
```


ROC curve for Ideal classifier (thr=0.5)
1.0
0.8
0.6
0.4
0.2
0.0
tpr
0.0
0.2
0.4
0.6
0.8
1.0
fpr
Ideal classifier (thr=0.5) auc=1.0
reference curve




```
              precision    recall  f1-score   support

       False       1.00      0.53      0.69      9508
        True       0.10      1.00      0.18       492

    accuracy                           0.55     10000
   macro avg       0.55      0.76      0.43     10000
weighted avg       0.96      0.55      0.66     10000
```


ROC curve for Ideal classifier (thr=0.95)
tpr
fpr
0.0
0.2
0.4
0.6
0.8
1.0
Ideal classifier (thr=0.95) auc=1.0
reference curve



	precision	recall	f1-score	support
False	1.00	1.00	1.00	9508
True	1.00	1.00	1.00	492
accuracy			1.00	10000
macro avg	1.00	1.00	1.00	10000
weighted avg	1.00	1.00	1.00	10000

Discussion For Part C (c)

Ideal Classifier (thr=0.5)

The ROC curve shows perfect separation (AUC=1.0), but the classification report reveals poor precision and recall for the positive class, demonstrating the need for an appropriate threshold in imbalanced datasets.

Ideal Classifier (thr=0.95)

Both the ROC curve and classification report show perfect performance with all metrics at 1.0, illustrating how the correct threshold can optimize classifier performance in identifying positive instances.