HW 4

Due: Nov 5th 11:59 PM

Total points: 80

Name(s) and EID(s):

If you prefer, you can work in groups of two. **Please note that only one student per team needs to submit the assignment but make sure to include both students' names and UT EIDs.**

For any question that requires a handwritten solution, you may upload scanned images of your solution in the notebook or attach them to the assignment . You may write your solution using markdown as well.

Please make sure your code runs and the graphs and figures are displayed in your notebook and PDF before submitting.

Q1. (10 points) Binary Classifier

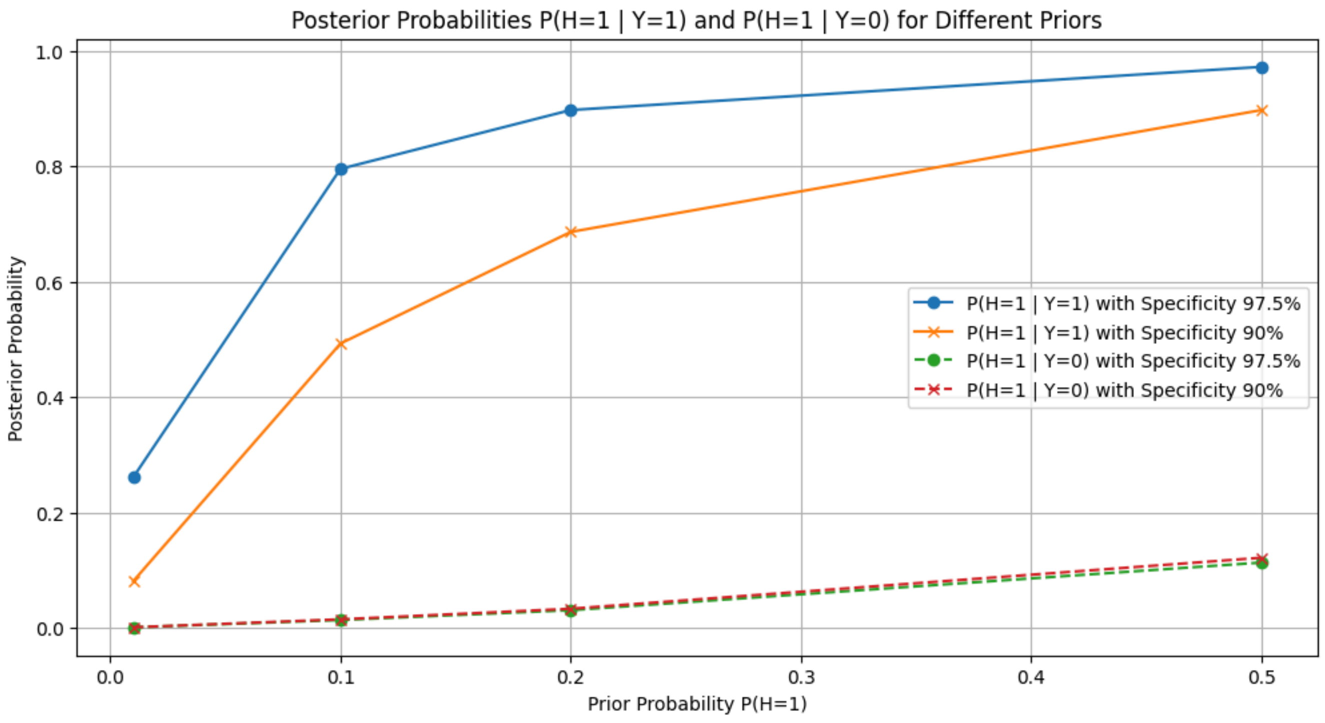
Consider the Covid-test scenario, where we have a test output Y = $\{0, 1\}$ and the true value H = $\{0, 1\}$. We also know the sensitivity (ie, $\mathbb{P}(Y = 1|H = 1)$) to be 87.5% and specificity (ie, $\mathbb{P}(Y = 0|H = 0)$) to be 97.5%. However, the prevalence of Covid in the area of interest (ie, the prior) $\mathbb{P}(H = 1)$ is not precisely known, and hence the below question:

a. (5 points) Calculate $\mathbb{P}(H = 1|Y = 1)$ and $\mathbb{P}(H = 1|Y = 0)$ for different values of prior $\mathbb{P}(H = 1)$ = $\{0.01, 0.1, 0.2, 0.5\}$. Plot them w.r.t the different values of prior.

b. (5 points) There is a cheaper test, with same sensitivity (ie, detects Covid +ve at the same rate) but with a reduced specificity of 90%(ie, tells healthy people that they are +ve, with a higher rate). Hence plot the same chart as in part a. with the new value of specificity.

```
import numpy as np
import matplotlib.pyplot as plt

# Sensitivity and Specificities
sensitivity = 0.875  # P(Y=1 | H=1)
specificity_1 = 0.975  # P(Y=0 | H=0) for the original test
specificity_2 = 0.90   # P(Y=0 | H=0) for the cheaper test

# Different prior probabilities for P(H=1)
priors = np.array([0.01, 0.1, 0.2, 0.5])

# Function to calculate P(H=1 | Y=1) and P(H=1 | Y=0)
def calculate_probabilities(sensitivity, specificity, prior):
    # P(Y=1) = P(Y=1 | H=1) * P(H=1) + P(Y=1 | H=0) * P(H=0)
    p_y_1 = sensitivity * prior + (1 - specificity) * (1 - prior)
    #print("prior",prior,p_y_1,sensitivity,specificity)

    # P(Y=0) = P(Y=0 | H=1) * P(H=1) + P(Y=0 | H=0) * P(H=0)
    p_y_0 = (1 - sensitivity) * prior + specificity * (1 - prior)

    # P(H=1 | Y=1) = P(Y=1 | H=1) * P(H=1) / P(Y=1)
    p_h_1_given_y_1 = (sensitivity * prior) / p_y_1

    # P(H=1 | Y=0) = P(Y=0 | H=1) * P(H=1) / P(Y=0)
    p_h_1_given_y_0 = ((1 - sensitivity) * prior) / p_y_0

    return p_h_1_given_y_1, p_h_1_given_y_0

# Calculate probabilities for each prior for both tests
results_specificity_1 = [calculate_probabilities(sensitivity, specificity_1, p) for p in priors]
results_specificity_2 = [calculate_probabilities(sensitivity, specificity_2, p) for p in priors]

# Extract probabilities for plotting
p_h_1_given_y_1_spec1 = [res[0] for res in results_specificity_1]
p_h_1_given_y_0_spec1 = [res[1] for res in results_specificity_1]

p_h_1_given_y_1_spec2 = [res[0] for res in results_specificity_2]
p_h_1_given_y_0_spec2 = [res[1] for res in results_specificity_2]

# Plot the results
plt.figure(figsize=(12, 6))

# Plot for P(H=1 | Y=1)
```

```
plt.plot(priors, p_h_1_given_y_1_spec1, marker='o', label='P(H=1 | Y=1) with Specificity 97.5%')
plt.plot(priors, p_h_1_given_y_1_spec2, marker='x', label='P(H=1 | Y=1) with Specificity 90%')

# Plot for P(H=1 | Y=0)
plt.plot(priors, p_h_1_given_y_0_spec1, marker='o', linestyle='--', label='P(H=1 | Y=0) with Specificity 97.5%')
plt.plot(priors, p_h_1_given_y_0_spec2, marker='x', linestyle='--', label='P(H=1 | Y=0) with Specificity 90%')

# Labeling the plot
plt.xlabel("Prior Probability P(H=1)")
plt.ylabel("Posterior Probability")
plt.title("Posterior Probabilities P(H=1 | Y=1) and P(H=1 | Y=0) for Different Priors")
plt.legend()
plt.grid(True)
plt.show()
```



Q2. Classification with Skorch (25 points)

In this question we will train and evaluate a simple neural network on a classification dataset using the skorch library. skorch is a scikit-learn compatible neural network library that wraps PyTorch. For more details on how to use the library see here. For this question, we will use a phishing dataset which consists of 11430 URLs with 87 extracted features. The dataset is already preprocessed and provided to you in the file `phishing_dataset.pt`. The dataset is balanced i.e. it contains 50% phishing and 50% legitimate URLs.

```
!pip install skorch mpld3
```

```
Collecting skorch
  Downloading skorch-1.0.0-py3-none-any.whl.metadata (11 kB)
Collecting mpld3
  Downloading mpld3-0.5.10-py3-none-any.whl.metadata (5.1 kB)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from skorch) (1.26.4)
Requirement already satisfied: scikit-learn>=0.22.0 in /usr/local/lib/python3.10/dist-packages (from skorch) (1.5.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from skorch) (1.13.1)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.10/dist-packages (from skorch) (0.9.0)
Requirement already satisfied: tqdm>=4.14.0 in /usr/local/lib/python3.10/dist-packages (from skorch) (4.66.6)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from mpld3) (3.1.4)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from mpld3) (3.8.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.0->skor
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->mpld3) (3.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (1.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (4.
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (1.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (24.1
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (10.4.0
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3) (3.2
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->mpld3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotli
Downloading skorch-1.0.0-py3-none-any.whl (239 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 239.4/239.4 kB 2.6 MB/s eta 0:00:00
Downloading mpld3-0.5.10-py3-none-any.whl (202 kB)
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 202.6/202.6 kB 11.6 MB/s eta 0:00:00
Installing collected packages: skorch, mpld3
Successfully installed mpld3-0.5.10 skorch-1.0.0
```

```
import torch
import numpy as np
import pandas as pd
from torch import nn
from skorch import NeuralNetClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, auc, RocCurveDisplay, ConfusionMatrixDisplay, accuracy_score, class
import matplotlib.pyplot as plt
```

```
from google.colab import files
uploaded = files.upload()
```

Choose files phishing_dataset_final.pt

- **phishing_dataset_final.pt**(n/a) - 6649652 bytes, last modified: 29/10/2024 - 100% done

```
Saving phishing_dataset_final.pt to phishing_dataset_final.pt
```

```
X_train, y_train, X_train_imbalanced, y_train_imbalanced, X_test, y_test, X_test_imbalanced, y_test_imbalanced = torch.load(
```

```
<ipython-input-5-6c5dc452fbe8>:1: FutureWarning: You are using `torch.load` with `weights_only=False` (the current defau
  X_train, y_train, X_train_imbalanced, y_train_imbalanced, X_test, y_test, X_test_imbalanced, y_test_imbalanced = torch
```

Part 1. (9 points)

In this part we will define our neural network. When building our network we will take as input the following:

1. inp_size: the number of input features
2. hidden_sizes: list of the size of each hidden layer in the network. Note that this does not include the size of the input and output layer. Eg: `[16, 8]` in which case your NN will look like: input_layer -> hidden_layer_1 (size 16) -> hidden_layer_2 (size 8) -> output_layer.
3. num_classes: the number of output classes which is equivalent to the size of the output layer. In our example, we only have 2 classes (phishing, non-phishing).
4. nonlin: the activation function. Eg: `torch.nn.ReLU()` .

The above arguments will be used to initialize our neural network. Notice that we want to be able to initialize an arbitrary size network specified by the hidden_sizes. To do this you must first initialize a `nn.ModuleList()` and append your layer followed by the activation for each specified size. You can refer to this for more help.

You must also define a `forward(X)` function which handles the forward pass of your network. Here `X` is your input and you must return the softmax output tensor from your network in this function.

```
class MyModule(nn.Module):
    def __init__(self, inp_size, hidden_sizes, num_classes=2, nonlin=nn.ReLU()):
        super().__init__()
        # define your hidden layers (self.hidden) as a nn.ModuleList() and append your nn.Linear layers based on the hidden_
        ##  START CODE  ##
        self.hidden = nn.ModuleList()

        self.hidden.append(nn.Linear(inp_size, hidden_sizes[0]))
        self.hidden.append(nonlin)

        # Define subsequent hidden layers
        for i in range(1, len(hidden_sizes)):
            self.hidden.append(nn.Linear(hidden_sizes[i - 1], hidden_sizes[i]))
            self.hidden.append(nonlin)

        # Define last hidden layer to output layer
        self.hidden.append(nn.Linear(hidden_sizes[-1], num_classes))

        ##  END CODE  ##

        # define softmax
        ##  START CODE  ## (1 line of code)
        self.softmax = nn.Softmax(dim=1)
        ##  END CODE  ##

    def forward(self, X):

        # calculate the output from your hidden layers
        # Hint: if your hidden layers are in the form of nn.ModuleList(),
        #       you must write a for loop to do a forward pass on all layers in the list
        ##  START CODE  ##
        for layer in self.hidden:
            X = layer(X)
        ##  END CODE  ##

        ##  END CODE  ##

        # calculate softmax on the output
        ##  START CODE  ## (1 line of code)
        out = self.softmax(X)
        ##  END CODE  ##
        return out
```

Part 2. (5 points)

In this part we will train and evaluate the neural network on our dataset. Use `X_train` and `y_train` tensors to train the model. Use `X_test` and `y_test` tensors to evaluate. **To get full credit you must obtain an accuracy of 95% or more on the test set.** You must use GridSearchCV from sklearn to search the best hyperparameters. Search over atleast 2 values (can be anything of your choice) of the following hyperparams:

1. Learning rate
2. Number of training epochs
3. hidden_sizes which is the input to your neural network you defined above
4. nonlin which is the activation function input to your neural network you defined above

Finally, use sklearn's ConfusionMatrixDisplay and RocCurveDisplay to caluclate and display the confusion matrix and ROC on the test set for the best model obtained using the grid search. Also use classification_report to calculate and print the precision and recall values of the positive and negative label.

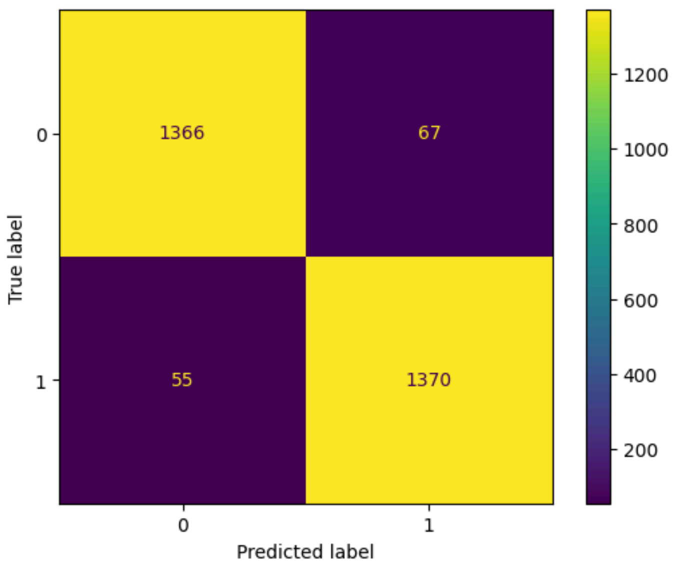
```
# define a NeuralNetClassifier() with batch size 256, torch.optim.Adam optimizer and torch.nn.CrossEntropyLoss as the criter
# IMPORTANT: also use iterator_train__shuffle=True to shuffle the training data during the training process
##  START CODE  ##
net = NeuralNetClassifier(
    MyModule,
    max_epochs=10,
    lr=0.01,
    batch_size=256,
    optimizer=torch.optim.Adam,
    criterion=torch.nn.CrossEntropyLoss,
    iterator_train__shuffle=True,
    module__inp_size=X_train.shape[1]
)
##  END CODE  ##

# deactivate skorch-internal train-valid split and verbose logging
net.set_params(train_split=False, verbose=0)

# define the parameters you want to search over as a dict
##  START CODE  ##
params = {
    'lr': [0.001, 0.01],                       # Learning rate options
    'max_epochs': [10, 20],                    # Number of epochs
    'module__hidden_sizes': [[16, 8], [32]], # Different hidden layer configurations
    'module__nonlin': [nn.ReLU(), nn.Tanh()] # Different activation functions
}
##  END CODE  ##

# define your GridSearchCV()
# IMPORTANT: use cv=3, scoring='accuracy' (to obtain best model based on accuracy)
# and refit=True (to retrain the model using the best hyperparams for later use i.e. evaluation).
##  START CODE  ## (1 line of code)
gs = GridSearchCV(net, params, cv=3, scoring='accuracy', refit=True)
##  END CODE  ##


# train your model
##  START CODE  ## (1 line of code)
gs.fit(X_train, y_train)
##  END CODE  ##

# print best params
print("best score: {:.3f}, best params: {}".format(gs.best_score_, gs.best_params_))
```

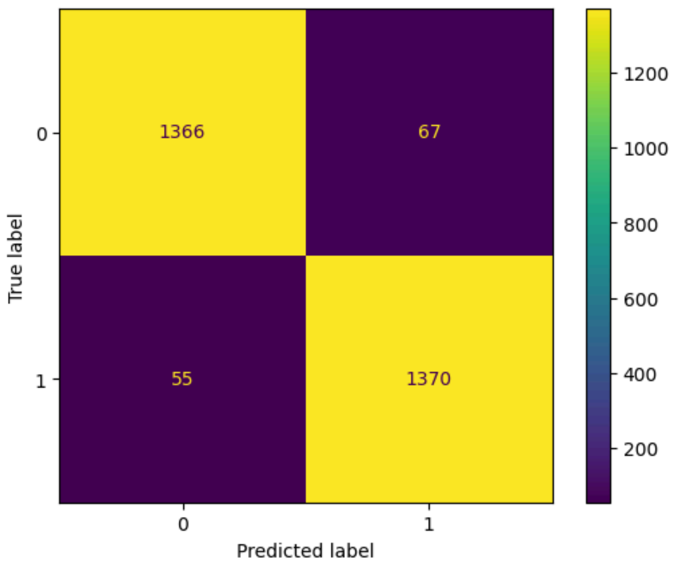
```
best score: 0.956, best params: {'lr': 0.01, 'max_epochs': 20, 'module__hidden_sizes': [32], 'module__nonlin': ReLU()}
```

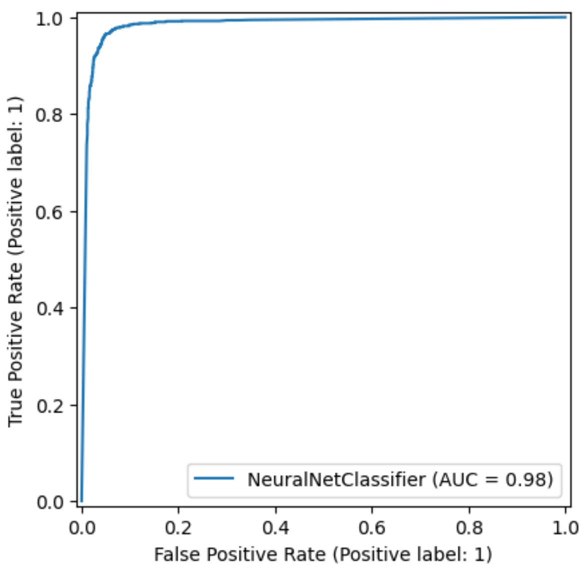
```
# Get the best model from your GridSearchCV object.
##  START CODE  ## (1 line of code)
net = gs.best_estimator_
```

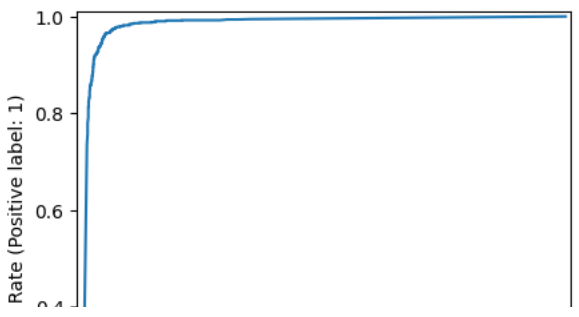
```
    ##  END CODE  ##

    # get predictions on the test data
    ##  START CODE  ## (1 line of code)
    y_pred_test = net.predict(X_test)
    ##  END CODE  ##

    # calculate accuracy and print
    print("Best Accuracy on Test: ", accuracy_score(y_test, y_pred_test))

    # calculate confusion matrix and use ConfusionMatrixDisplay to display the matrix
    ##  START CODE  ##
    confusion_mat = ConfusionMatrixDisplay.from_predictions(y_test, y_pred_test)
    confusion_mat.plot()
    ##  END CODE  ##

    # use RocCurveDisplay to display the roc curve and AUC
    ##  START CODE  ## (1 line of code)
    RocCurveDisplay.from_estimator(net, X_test, y_test).plot()
    ##  END CODE  ##

    # print sklearn's classification_report which consists of precision and recall for both labels
    ##  START CODE  ## (1 line of code)
    print(classification_report(y_test, y_pred_test))
    ##  END CODE  ##
```

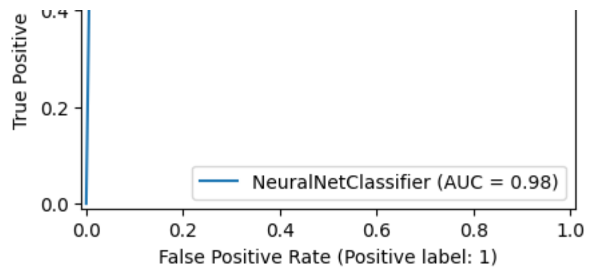
```
Best Accuracy on Test:  0.9573128061581525
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7d6663f2fb50>
```











Part 3. (5 points)

In this part we will deal with a more realistic scenario. In reality most emails we receieve are non-phishing legitimate emails. Hence in a real world dataset, we can expect that there will be way more datapoints with a negative label than positive. Hence the dataset will be imbalanced.

Use `X_train_imbalanced` and `y_train_imbalanced` tensors to train the model. Use `X_test_imbalanced` and `y_test_imbalanced` tensors to evaluate. **To get full credit you must obtain an accuracy of 95% or more on the imbalanced test set.** You must use GridSearchCV from sklearn to search the best hyperparameters. Search over atleast 2 values (can be anything of your choice) of the following hyperparams:

1. Learning rate
2. Number of training epochs
3. hidden_sizes which is the input to your neural network you defined above
4. nonlin which is the activation function input to your neural network you defined above

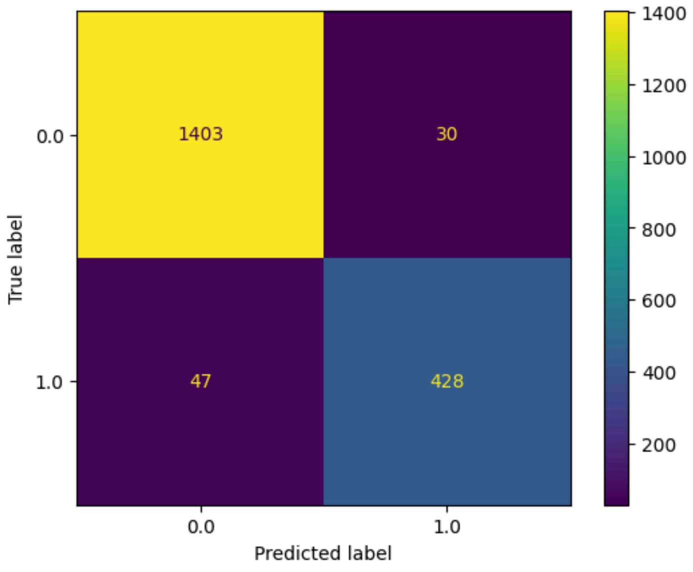
Finally, use sklearn's ConfusionMatrixDisplay and RocCurveDisplay to caluclate and display the confusion matrix and ROC on the imbalanced test set for the best model obtained using the grid search. Also use classification_report to calculate and print the precision and recall values of the positive and negative label.

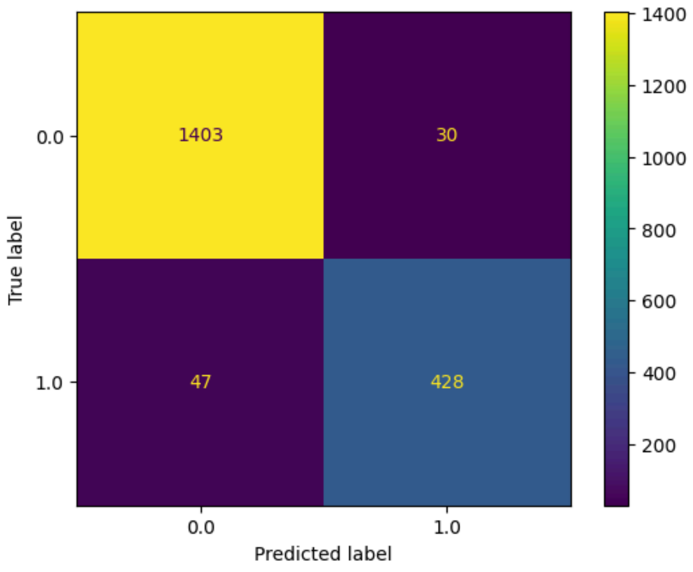
```
# define a NeuralNetClassifier() with batch size 256, torch.optim.Adam optimizer and torch.nn.CrossEntropyLoss as the criter
# IMPORTANT: also use iterator_train__shuffle=True to shuffle the training data during the training process
##  START CODE  ##
net_imbalanced = NeuralNetClassifier(
    MyModule,
    max_epochs=10,
    lr=0.01,
    batch_size=256,
    optimizer=torch.optim.Adam,
    criterion=torch.nn.CrossEntropyLoss,  # Use CrossEntropyLoss for multi-class classification
    iterator_train__shuffle=True,
    module__inp_size=X_train_imbalanced.shape[1]
)
##  END CODE  ##


# deactivate skorch-internal train-valid split and verbose logging
net_imbalanced.set_params(train_split=False, verbose=0)

# define the parameters you want to search over as a dict
##  START CODE  ##
params = {
    'lr': [0.001, 0.01],                      # Learning rate options
    'max_epochs': [10, 20],                   # Number of epochs
    'module__hidden_sizes': [[16, 8], [32]], # Different hidden layer configurations
    'module__nonlin': [nn.ReLU(), nn.Tanh()] # Different activation functions
}
##  END CODE  ##

# define your GridSearchCV()
# IMPORTANT: use cv=3, scoring='accuracy' (to obtain best model based on accuracy)
# and refit=True (to retrain the model using the best hyperparams for later use i.e. evaluation).
##  START CODE  ## (1 line of code)
gs_imbalanced = GridSearchCV(net_imbalanced, params, cv=3, scoring='accuracy', refit=True)
##  END CODE  ##


# train your model on the imbalanced train dataset
##  START CODE  ## (1 line of code)
gs_imbalanced.fit(X_train_imbalanced, y_train_imbalanced)
##  END CODE  ##

# print best params
print("best score: {:.3f}, best params: {}".format(gs_imbalanced.best_score_, gs_imbalanced.best_params_))
```

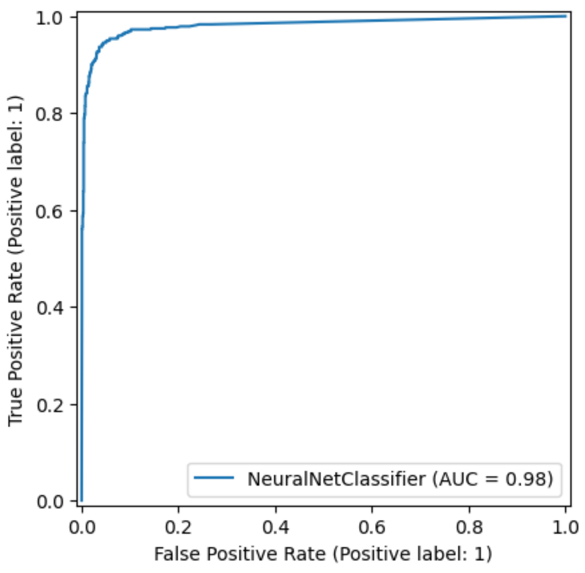
```
best score: 0.960, best params: {'lr': 0.01, 'max_epochs': 20, 'module__hidden_sizes': [32], 'module__nonlin': ReLU()}
```

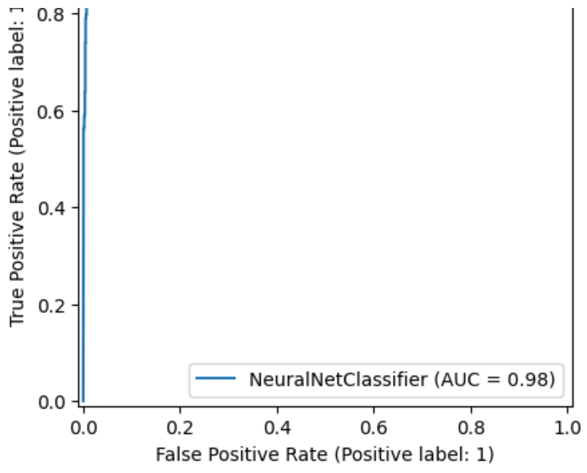
```
    # Get the best model from your GridSearchCV object.
    ##  START CODE  ## (1 line of code)
    net_imbalanced = gs_imbalanced.best_estimator_
    ##  END CODE  ##

    # get predictions on the imbalanced test data
    ##  START CODE  ## (1 line of code)
    y_pred_test = net_imbalanced.predict(X_test_imbalanced)
    ##  END CODE  ##

    # calculate accuracy and print
    print("Best Accuracy on Imbalanced Test: ", accuracy_score(y_test_imbalanced, y_pred_test))

    # calculate confusion matrix and use ConfusionMatrixDisplay to display the matrix
    ##  START CODE  ##
    confusion_mat = ConfusionMatrixDisplay.from_predictions(y_test_imbalanced, y_pred_test)
    confusion_mat.plot()
    ##  END CODE  ##

    # use RocCurveDisplay to display the roc curve and AUC
    ##  START CODE  ## (1 line of code)
    RocCurveDisplay.from_estimator(net_imbalanced, X_test_imbalanced, y_test_imbalanced).plot()
    ##  END CODE  ##

    # print sklearn's classification_report which consists of precision and recall for both labels
    ##  START CODE  ## (1 line of code)
    print(classification_report(y_test_imbalanced, y_pred_test))
    ##  END CODE  ##
```

```
Best Accuracy on Imbalanced Test:  0.959643605870021
              precision    recall  f1-score   support

         0.0       0.97      0.98      0.97      1433
         1.0       0.93      0.90      0.92       475

    accuracy                           0.96      1908
   macro avg       0.95      0.94      0.95      1908
weighted avg       0.96      0.96      0.96      1908
```









```
# do not modify anything here
from sklearn import metrics
%matplotlib inline
import mpld3
mpld3.enable_notebook()

# IMPORTANT: to use this function simply call compare_roc_curves()
def compare_roc_curves():
  y_pred = net.predict_proba(X_test)[:, 1]
  fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
  auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
  plt.plot(fpr,tpr,label="Balanced Model, AUC="+str(auc))

  y_pred = net_imbalanced.predict_proba(X_test)[:, 1]
  fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
  auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
  plt.plot(fpr,tpr,label="Imbalanced Model, AUC="+str(auc))
  plt.legend()
```

Part 4. (6 points)

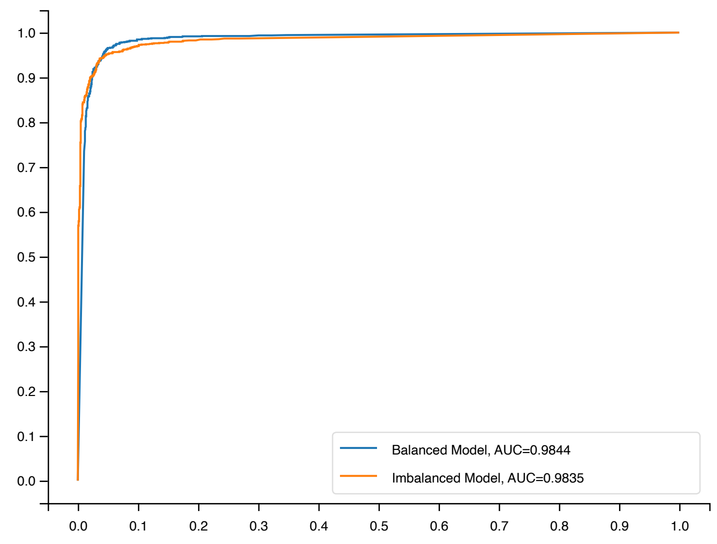
Briefly answer the following questions (1 paragraph should suffice):

a. How does the performance (wrt the ROC curve) of the model trained on the imbalanced dataset compare to that of the model trained on the balanced dataset? *(Hint: Use compare_roc_curves() function to plot both ROC curves on the same plot and zoom in on the curved part. Think about it indicates about the decision threshold of the 2 models)* (2 points)

b. Is accuracy the best metric to judge a model performance in this case? When dealing with a phishing dataset which set of metric(s) is most important to get a real sense of the model performance? *(Hint: think about which value in your confusion matrix matters the most and a metric related to that value)* (2 points)

c. What are some techniques to help counter label imbalance in your training set? (2 points)

```
compare_roc_curves()
```



(a) The ROC curves for both the balanced and imbalanced models are very similar, with both achieving high AUC scores (around 0.98). This indicates that both models are capable of distinguishing between classes effectively. However, zooming in on the curved part shows slight differences in how each model behaves at lower thresholds. The balanced model generally performs better across various thresholds, as it was trained on data that allowed it to learn the decision boundaries between classes more accurately. We can even observe this in Precison-Recall values for both balance and imbalanced -

For Balanced:

- Precision : 0.95
- Recall : 0.95

For Imbalnced:

- Precison : 0.93
- Recall : 0.89

In the balnced dataset model, we are able to achieve 0.95 TPR with 0.95 Precison, whereas using model trained on Imabalnced data we're able achieve only 0.89 TPR with a recall of 0.93 at same threshold

(b) In this case, accuracy is not the best metric to judge model performance due to the class imbalance in the dataset. For phishing detection, the key metrics to focus on are precision, recall, and especially recall (sensitivity) for the positive (phishing) class. High recall is essential because it indicates the model's ability to detect phishing emails correctly, reducing the number of false negatives (missed phishing emails)

(c) Here are some of the techniques we can use to counter label imbalance

- Class Weights: Assign higher weights to the minority class in the loss function, which penalizes the model more for misclassifying phishing emails.
- Resampling: Use techniques like oversampling (e.g., SMOTE) to increase the number of minority class samples or undersampling to reduce the majority class samples

Q3. Logistic Regression and Softmax Classifier (10 points)

Part 1. (5 points)

We have a dataset $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{1, \dots, K\}$ for all i. Suppose we have a K-way softmax classifier:

$$P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x} + b_k)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x} + b_j)}$$

where $\mathbf{w}_k \in \mathbb{R}^d$ and $b_k \in \mathbb{R}$ are the weight and bias parameters from the weight matrix and bias vector:

$$\mathbf{W} = \begin{pmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \vdots \\ \mathbf{w}_K^\top \end{pmatrix}$$

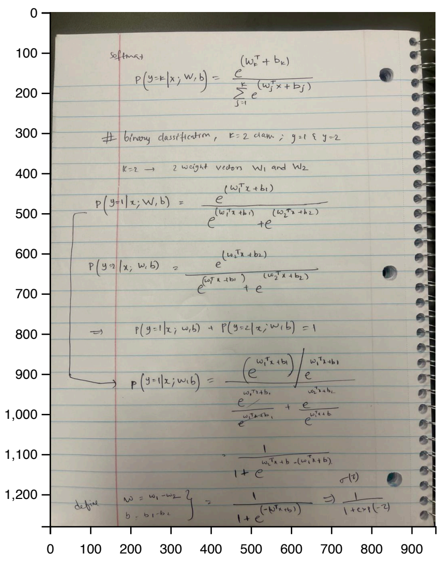
$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{pmatrix}$$

Show that when $K = 2$, this softmax classifier is equivalent to logistic regression.

```
from google.colab import files
uploaded = files.upload()
```

```
Choose files 2 files
• H4_1.jpeg(image/jpeg) - 114510 bytes, last modified: 04/11/2024 - 100% done
• H4_2.jpeg(image/jpeg) - 99542 bytes, last modified: 04/11/2024 - 100% done
Saving H4_1.jpeg to H4_1 (2).jpeg
Saving H4_2.jpeg to H4_2 (2).jpeg
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load the images
img1 = mpimg.imread('H4_1 (2).jpeg')
img2 = mpimg.imread('H4_2 (2).jpeg')

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Display the first image in the first subplot
axes[0].imshow(img1)
axes[0].axis('off')  # Optional: Turn off axis for the first image

# Display the second image in the second subplot
axes[1].imshow(img2)
axes[1].axis('off')  # Optional: Turn off axis for the second image

# Display the figure
plt.show()
```


0
100
200
300
400
500
600
700
800
900
1,000
1,100
1,200

softmat

$$P(y=k|x;W,b) = \frac{e^{(w_k^T + b_k)}}{\sum_{j=1}^{k} e^{(w_j^T x + b_j)}}$$

# binary classification, $k=2$ class ; $y=1$ & $y=2$

$k=2 \rightarrow$ 2 weight vectors W_1 and W_2

$$P(y=1|x;W,b) = \frac{e^{(w_1^T x + b_1)}}{e^{(w_1^T x + b_1)} + e^{(w_2^T x + b_2)}}$$

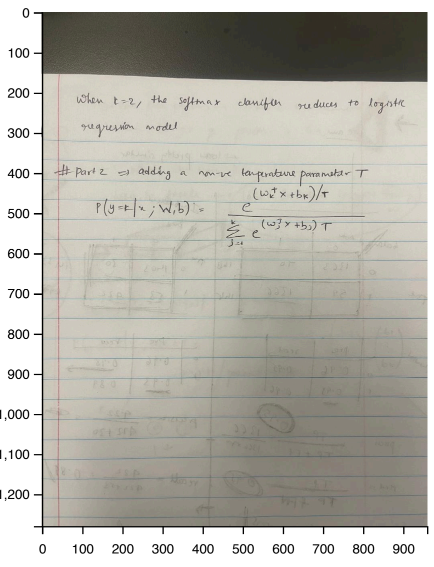
$$P(y=2|x;w,b) = \frac{e^{(w_2^T x + b_2)}}{e^{(w_1^T x + b_1)} + e^{(w_2^T x + b_2)}}$$

$\Rightarrow \quad P(y=1|x;w,b) + P(y=2|x;w,b) = 1$

$$P(y=1|x;w,b) = \frac{\left(e^{w_1^T x + b_1}\right) / e^{w_1^T x + b_1}}{\frac{e^{w_1^T x + b_1}}{e^{w_1^T x + b_1}} + \frac{e^{w_2^T x + b_2}}{e^{w_1^T x + b_1}}}$$

$$= \frac{1}{1 + e^{w_2^T x + b - (w_1^T x + b)}}$$

define $\left. \begin{array}{l} w = w_1 - w_2 \\ b = b_1 - b_2 \end{array} \right\} = \frac{1}{1 + e^{(-(w^T x + b))}} \Rightarrow \overset{\sigma(z)}{\frac{1}{1 + \exp(-z)}}$


0
100
200
300
400
500
600
700
800
900
1,000
1,100
1,200

When $k=2$, the softmax classifier reduces to logistic regression model

# part 2 $\Rightarrow$ adding a non-ve temperature parameter T

$$P(y=k|x;W,b) = \frac{e^{(w_k^T x + b_k)/T}}{\sum_{j=1}^{k} e^{(w_j^T x + b_j)T}}$$

Part 2. (5 points)

Suppose we slightly tweak the softmax classifier by adding an additional non-negative temperature parameter T:

$$P(y = k|\mathbf{x}; \mathbf{W}, \mathbf{b}) = \frac{\exp((\mathbf{w}_k^\top \mathbf{x} + b_k)/T)}{\sum_{j=1}^{K} \exp((\mathbf{w}_j^\top \mathbf{x} + b_j)/T)}$$

(a) What will happen to the classifier output when $T \rightarrow \infty$? How about when $T \rightarrow 0$?

(b) How is this parameter useful for our model and how do we choose it?

Answer

(a)

- As T approasches Infinity, the exponent values get to zero and exp of 0 is 1. This forces the probabilities of all the k classes to be same P = 1/k.
- As T approaches zero, the exponents become very large in magnitude. The softmax function will amplify the differences between the values of exp(wx + b) such that the largest term in the exponent dominates the sum in the denominator thus, the probability for highest exp(wx+b) will approach 1, while all other probabilities approach 0. Therefore, when T→0, the softmax output becomes a "hard" or deterministic classification, with the probability concentrating entirely on the class with the highest score

(b) The temperature parameter T allows us to control the "confidence" or "sharpness" of the model's predictions:

- Higher T makes the output probabilities more "soft," meaning the model is less confident in its predictions and distributes the probability more evenly across classes.
- Lower T makes the output probabilities more "sharp," making the model more confident in its predictions. For very low T, the model will output high probabilities for a single class and very low probabilities for others, similar to a hard classification.

Applications of the Temperature Parameter:

- Model Calibration: In certain applications, we may want a model that does not give overly confident predictions, especially when dealing with noisy or uncertain data. Increasing T can help produce smoother, more calibrated probabilities that better reflect uncertainty.

Question 4 : Ensemble Methods for Classification (25 pts)

In this question, we will compare the performances of different ensemble methods for classification problems: Bagging, Random Forest and XGBoost classifiers.

We will look at the GiveMeSomeCredit dataset for this question. The dataset is extremely large so for this question we will only consider a subset which has been provided along with the notebook for this assignment. The dataset has already been split into train and test sets.

The task is to predict the probability that someone will experience financial distress in the next two years.

```
# Only use this code block if you are using Google Colab.
# If you are using Jupyter Notebook, please ignore this code block. You can directly upload the file to your Jupyter Noteboc
from google.colab import files

## It will prompt you to select a local file. Click on “Choose Files” then select and upload the file.
## Wait for the file to be 100% uploaded. You should see the name of the file once Colab has uploaded it.
uploaded = files.upload()
```

Choose files credit_data_new.csv

- **credit_data_new.csv**(text/csv) - 234084 bytes, last modified: 29/10/2024 - 100% done

```
Saving credit_data_new.csv to credit_data_new.csv
```

```
import pandas as pd

data = pd.read_csv('credit_data_new.csv')
data.drop(data.columns[data.columns.str.contains('unnamed',case = False)],axis = 1, inplace = True)
data.head()
```

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOf(
0	0	0.000978	84	0	0.000240	4166.0	
1	0	0.162015	59	0	0.227180	9300.0	
2	0	0.211747	58	0	0.550531	6500.0	
3	0	0.890781	56	0	810.000000	NaN	
4	0	0.698895	59	2	0.268481	11240.0	

Next steps: Generate code with data View recommended plots New interactive sheet

```
from sklearn.model_selection import train_test_split
y = data['SeriousDlqin2yrs']
X = data.drop(['SeriousDlqin2yrs'],axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 7)

print('train:',X_train.shape, y_train.shape)
print('test:',X_test.shape, y_test.shape)
```

```
train: (3750, 10) (3750,)
test: (1250, 10) (1250,)
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import (train_test_split,GridSearchCV)
from sklearn.metrics import (accuracy_score,roc_auc_score)
from sklearn.ensemble import (RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier)
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from time import time
import xgboost
%matplotlib inline
```

```
columns_list = list(X.columns)
```

a. (2.5 pts) Fit a Decision Tree Classifier with random_state = 14 for this classification problem. Report the accuracy_score and roc_auc_score on the test set.

```
def fit_classifier(clf):
  # Fit the classifier on the training set
  ### START CODE ###
  clf.fit(X_train, y_train)
  ### END CODE ###
  return clf


def evaluate_classifier(clf, X_test, y_test):
  # Compute the accuracy_score, and roc_auc_score on the test set
  ### START CODE ###
  y_pred = clf.predict(X_test)
  y_pred_proba = clf.predict_proba(X_test)[:, 1]  # Probability for the positive class


  acc_score = accuracy_score(y_test, y_pred)
  auc_score = roc_auc_score(y_test, y_pred_proba)
  ### END CODE ###
  print("Accuracy_score: {}, ROC_AUC_score: {}".format(acc_score, auc_score))


print("Decision Tree")
# Initialize your decision tree classifier
### START CODE ###
dt_clf = DecisionTreeClassifier(random_state=14)
### END CODE ###

dt_clf = fit_classifier(dt_clf)
evaluate_classifier(dt_clf, X_test, y_test)
```

```
Decision Tree
Accuracy_score: 0.9072, ROC_AUC_score: 0.5988347811560307
```

b. (2.5 pts) Create a Bagging of 25 classifiers (i.e, n_estimators=25) with random_state=14. Please use Decision Tree Classifier with random_state=14 as the base classifier. Report accuracy_score and roc_auc_score on the test data for this emsemble classifier.

```
print("Bagging of Decesion Trees")
# Initialize your bagging classifier
### START CODE ###
bag_clf = BaggingClassifier(estimator=DecisionTreeClassifier(random_state=14), n_estimators=25, random_state=14)
### END CODE ###

bag_clf = fit_classifier(bag_clf)
evaluate_classifier(bag_clf, X_test, y_test)
```

```
Bagging of Decesion Trees
Accuracy_score: 0.932, ROC_AUC_score: 0.7752829518877382
```

c. (5 pts) In this question, you will fit a Random Forest model on the training data for this classification task.

1. First, please find the best parameters (including *n_estimators*, *max_features* and *criterion*) using GridSearchCV. Report the optimal parameters obtained by GridSearch.
2. Fit a model using the best parameters, and report the confusion matrix and roc_auc_score on test data.

```
def grid_search_for_classifier(clf, param_grid, X_train, y_train):
  # Grid search
  grid_search = GridSearchCV(clf, param_grid=param_grid)
  # grid_search = GridSearchCV(clf, param_grid=param_grid, cv=3, scoring='roc_auc', n_jobs=-1)

  # Conduct grid search using the training set (1 line of code only)
  ### START CODE ###
  grid_search.fit(X_train, y_train)
  ### END CODE ###
  print(grid_search.best_params_)

  # Set the best paramters for your clf (1 line of code only)
  ### START CODE ###
  clf = grid_search.best_estimator_
  ### END CODE ###
  return clf


def train_and_evaluate_classifier(clf, X_train, y_train, X_test, y_test):
  t0 = time()
  # Fit your classifier on the training set
  ### START CODE ###
  clf.fit(X_train, y_train)
  ### END CODE ###
  print("training time", round(time()-t0, 3), "s")

  t0 = time()
  y_pred = clf.predict(X_test)
  print("predict time", round(time()-t0, 3), "s")

  print("Confusion matrix: ")
  # Print the confusion matrix computed from the test set (1 line of code only)
  ### START CODE ###
  print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
  ### END CODE ###

  y_pred_proba = clf.predict_proba(X_test)[:,1]
  acc_score = accuracy_score(y_test, y_pred)
  auc_score = roc_auc_score(y_test, y_pred_proba)

  print("Accuracy: {}, AUC_ROC: {}".format(acc_score, auc_score))
  return clf


param_grid = {"n_estimators": [1, 10, 50, 100],
              "max_features": [1, 5, 10, "sqrt"],
              "criterion": ['gini','entropy'],
              "random_state": [17]}

# Initialize your random forest classifier
### START CODE ###
rf_clf = RandomForestClassifier()
### END CODE ###
rf_clf = grid_search_for_classifier(rf_clf, param_grid, X_train, y_train)
train_and_evaluate_classifier(rf_clf, X_train, y_train, X_test, y_test)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
{'criterion': 'entropy', 'max_features': 1, 'n_estimators': 100, 'random_state': 17}
training time 0.465 s
predict time 0.027 s
Confusion matrix:
Confusion matrix:
 [[1162    6]
 [  77    5]]
Accuracy: 0.9336, AUC_ROC: 0.7859328015369195
```

```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_features=1, random_state=17)
```

d. (10 pts) This time, let us use XGBoost for the same task. Please find the best parameters (including *n_estimators, learning_rate*); fit your model using the best parameters, and report the confusion matrix and roc_auc_score on test data.

```
param_grid = {"n_estimators": [10, 100],
        "learning_rate": [0.01, 0.1, 0.5],
        "random_state": [17]
        }
```

```
# Initialize your XGBoost classifier
### START CODE ###
xgb_clf = xgboost.XGBClassifier()
### END CODE ###
xgb_clf = grid_search_for_classifier(xgb_clf, param_grid, X_train, y_train)
train_and_evaluate_classifier(xgb_clf, X_train, y_train, X_test, y_test)
```

```
{'learning_rate': 0.01, 'n_estimators': 100, 'random_state': 17}
training time 0.162 s
predict time 0.01 s
Confusion matrix:
Confusion matrix:
 [[1167    1]
 [  80    2]]
Accuracy: 0.9352, AUC_ROC: 0.7976528566655529
```

```
▾                          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, random_state=17, ...)
```

f. (5 pts) Compare the performance of decision tree from part a) with the ensemble methods. Briefly explain which of the three ensemble methods performed better and why?

Answer

The decision tree had an accuracy of 0.9072 and a ROC AUC of 0.5988. In contrast, all ensemble methods outperformed it: Bagging (accuracy: 0.9320, ROC AUC: 0.7753), Random Forest (accuracy: 0.9336, ROC AUC: 0.7859), and XGBoost (accuracy: 0.9352, ROC AUC: 0.7977).

Among these, XGBoost performed the best due to its boosting technique, which focuses on correcting errors, regularization to prevent overfitting, and efficient handling of missing values. This results in higher accuracy and better ROC AUC scores, indicating superior classification performance compared to the decision tree.

Q5. Ensembles Descriptive Questions (10 points)

Part 1. (5 points)

Gradient tree boosting iteratively adds regression trees into the ensemble model. In XGBoost, how are those individual trees learned and list three techniques used to prevent over-fitting. Briefly describe in your own words in less than 6 sentences. Please check Ch. 2 of this paper XGBoost: A Scalable Tree Boosting System for the details.

Answer

n XGBoost, individual trees in the ensemble model are learned sequentially, where each tree aims to correct the errors made by the previous trees. This is achieved by fitting each new tree to the residuals (the differences between the predicted values and the actual target values) of the ensemble's predictions. Three techniques used to prevent overfitting in XGBoost include:

Regularization: XGBoost incorporates L1 (Lasso) and L2 (Ridge) regularization to penalize complex models, discouraging overfitting by controlling the magnitude of the coefficients.

Early Stopping: During training, the model can be evaluated on a validation set, and training can be stopped when the model's performance ceases to improve, thus avoiding unnecessary complexity.

Subsampling: XGBoost randomly samples a portion of the training data for each iteration, which helps to reduce variance and makes the model less sensitive to noise in the dataset.

These techniques collectively enhance the generalization ability of the model, ensuring that it performs well on unseen data.

Part 2. (5 points)

List **two** challenges of training mixture of experts (MoE) and briefly explain each of them in less than 3 sentences.

Answer

Complexity of Expert Selection: In MoE, determining which expert to activate for a given input can be complex. The gating mechanism, responsible for selecting the appropriate expert, must effectively learn to route inputs based on their characteristics, which can require significant training data and computational resources.

Overfitting: Since MoE models can have many parameters due to the multiple experts, there is a risk of overfitting, especially if the number of training samples is limited. If the model becomes too complex, it may perform well on the training data but fail to generalize to unseen data, leading to poor performance in practice.

```
# a
# a
# a
# a
# a

# a
# a
# a
# a
# a

# a
# a
# a
# a

# a
# a
# a
# a


# a
```