Name:

Matthew Tang

Netid:

mhtang2

CS 441 - HW 4: Trees and MLPs

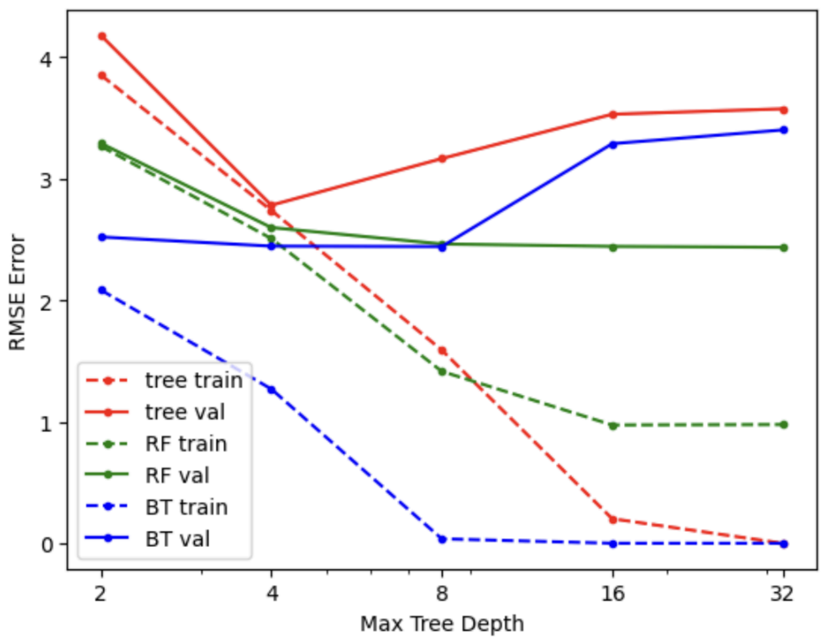
Complete the sections below. You do not need to fill out the checklist. **Do select all relevant pages in Gradescope.**

Total Points Claimed **[ ] / 170**

1. Model Complexity with Tree Regressors
    a. Depth vs. Error plot [ ] / 10
    b. Analysis [ ] / 20
2. MLPs with MNIST
    a. Loss Curves [ ] / 20
    b. Model Selection and Results [ ] / 20
3. Species Prediction
    a. Feature Analysis [ ] / 10
    b. Simple Rule [ ] / 10
    c. Model Design [ ] / 10
4. Stretch Goals
    a. Improve MNIST classification [ ] / 30
    b. A second simple rule [ ] / 10
    c. Positional encoding of RGB Image [ ] / 30

1. Model Complexity with Tree Regressors

a. Include your plot below.



b. Analyze your results:

1. For a given max tree depth, which of regressor model (single tree, random forest, boosted tree) has the lowest bias (or most powerful)?

Boosted Tree

2. For single regression trees, what tree depth achieves minimum validation error?

4

3. A model "overfits" when increasing the complexity increases the validation error. Which model is least prone to overfitting? Why?

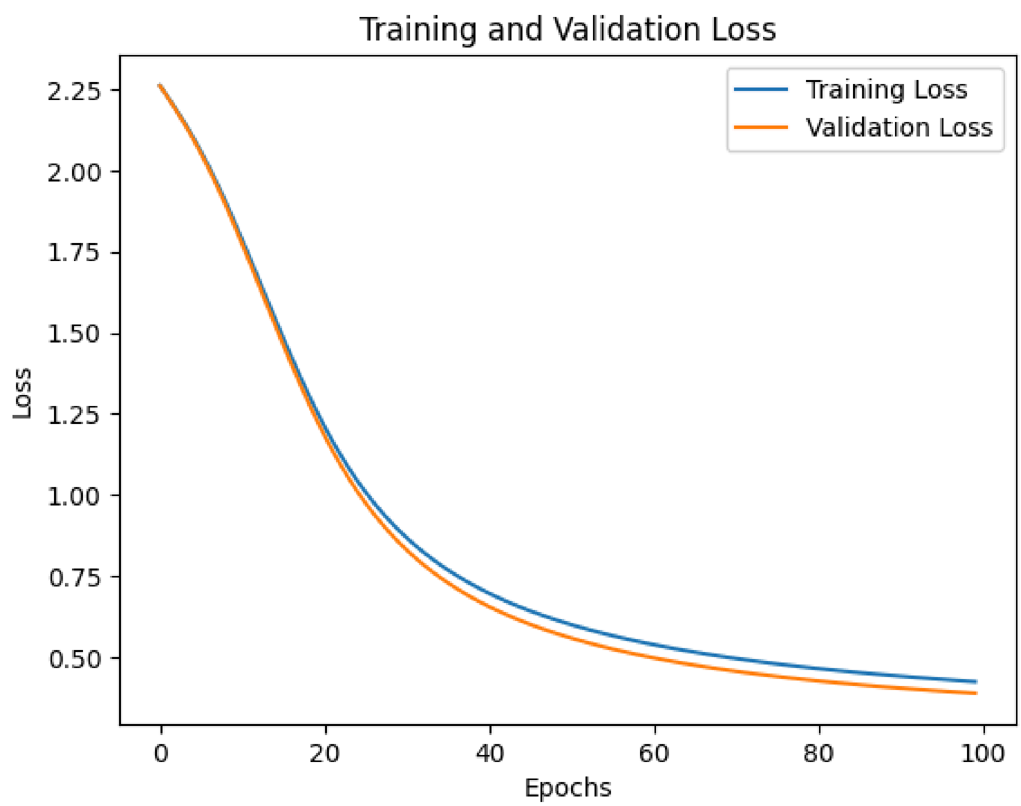
Random forests. Because it aggregates results from multiple independent models, variance is decreased. Also because it randomly selects a subset of features, so it is less likely to overfit.

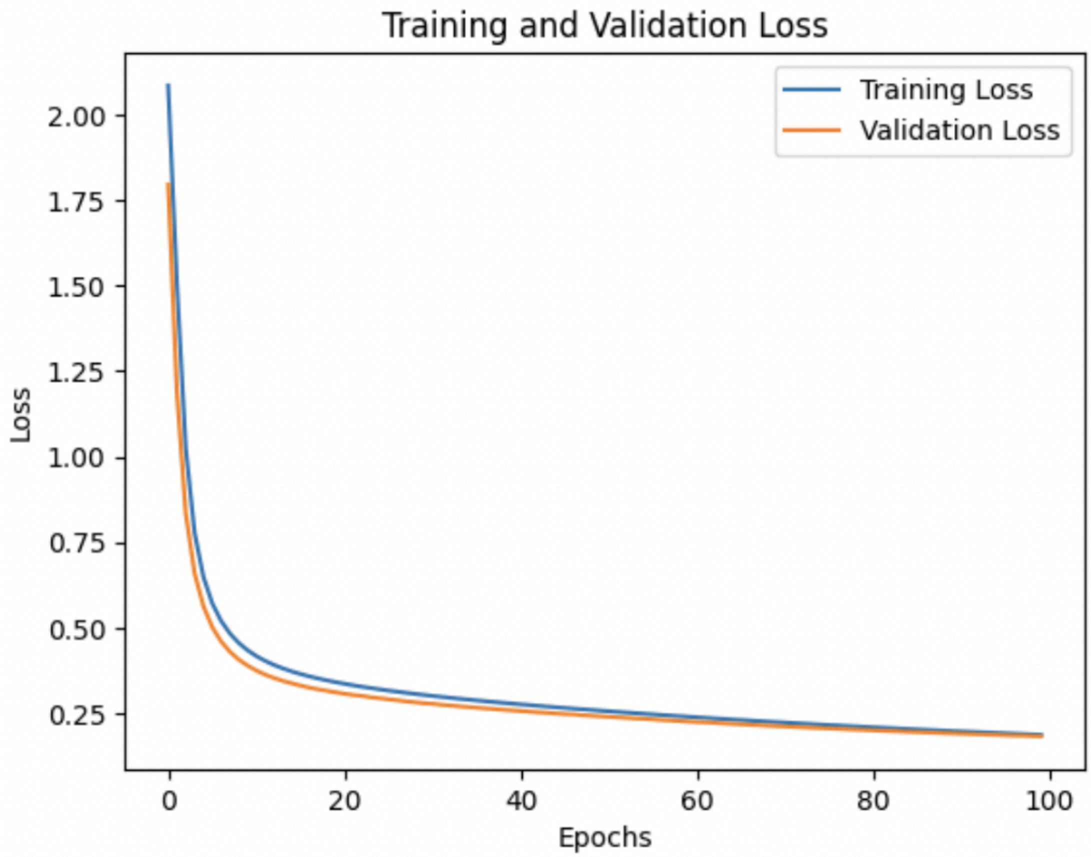
4. Do boosted trees seem to perform better with smaller or larger trees? Why?

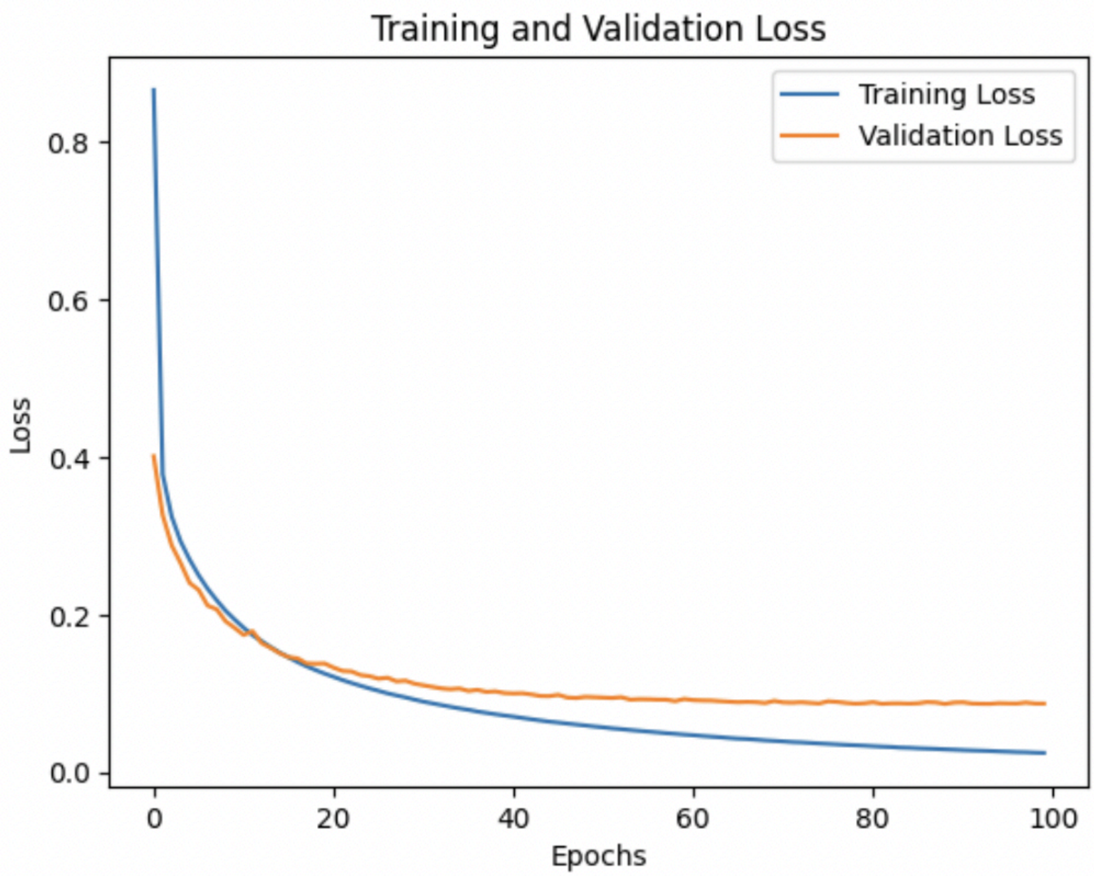
Small trees, since large trees will be more prone to overfit.

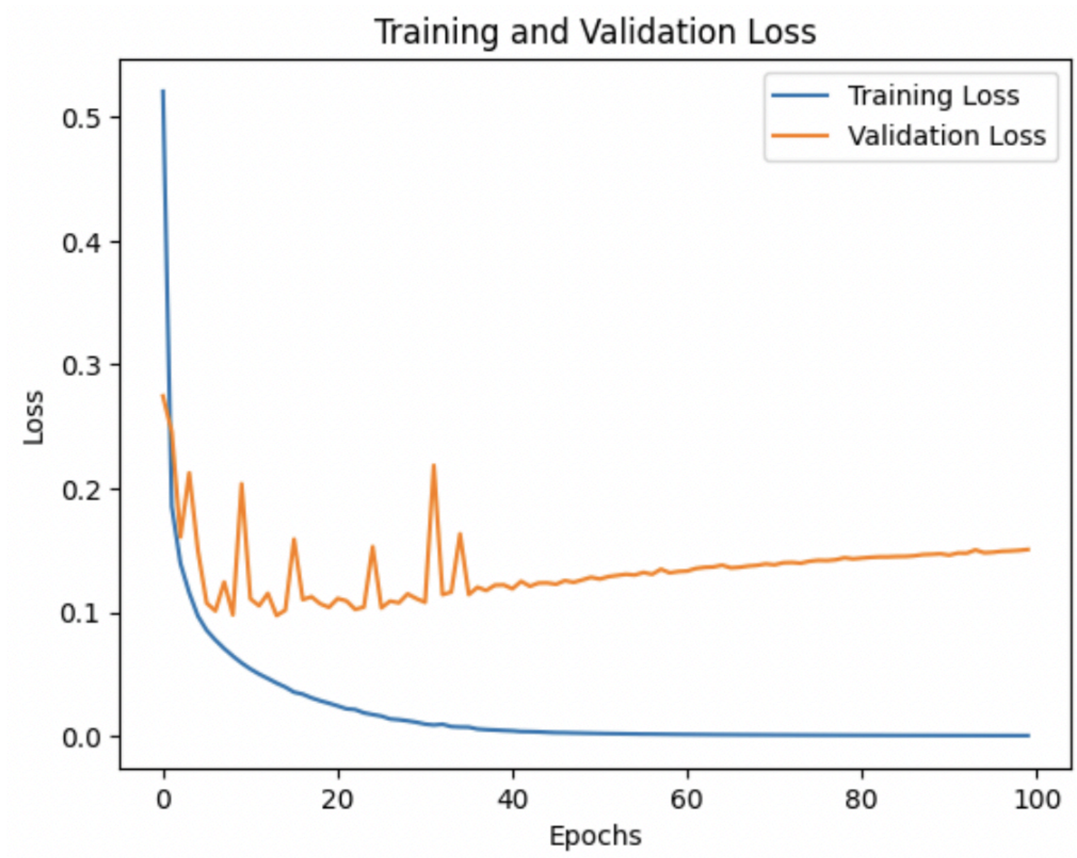
2. **MLPs with MNIST**

   a. **Show the loss curves** for 3 learning rates (1E-2, 1E-1, 1E1) training for 100 epochs. An example of the loss curves is shown for LR=0.001.

Learning Rate	Loss Curve Plot
0.001	

0.01	Training and Validation Loss Training Loss Validation Loss Loss Epochs 2.00 1.75 1.50 1.25 1.00 0.75 0.50 0.25 0 20 40 60 80 100

0.1	 Training and Validation Loss Training Loss Validation Loss Loss Epochs 0.0 0.2 0.4 0.6 0.8 0 20 40 60 80 100

1	

b. Model selection and results

Select the best hyperparameters (learning rate and number of epochs up to 100) based on minimizing the validation loss.

Learning Rate

0.1

Epochs

95

Report the losses and errors for the model trained with these hyperparameters:

Use scientific notation with one decimal place, e. 1.5E-3

Training **Loss**	Validation **Loss**	
2.6E-2	8.9E-2	

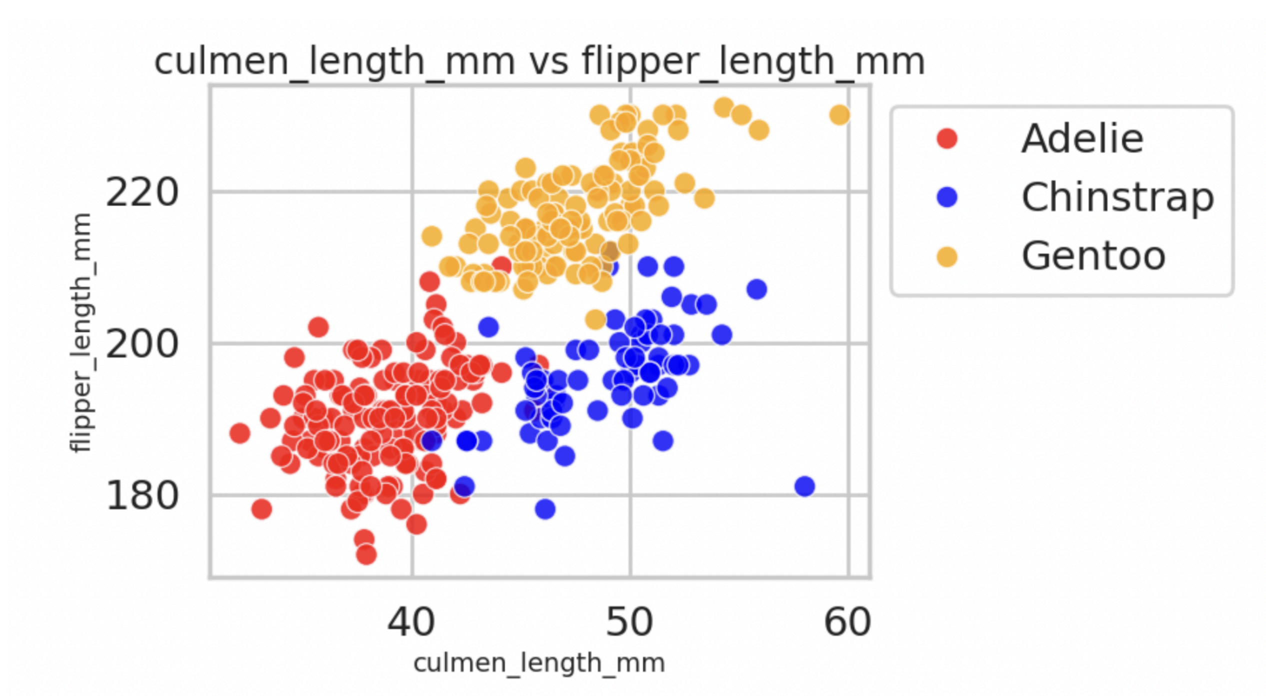
Show two decimal places for percent

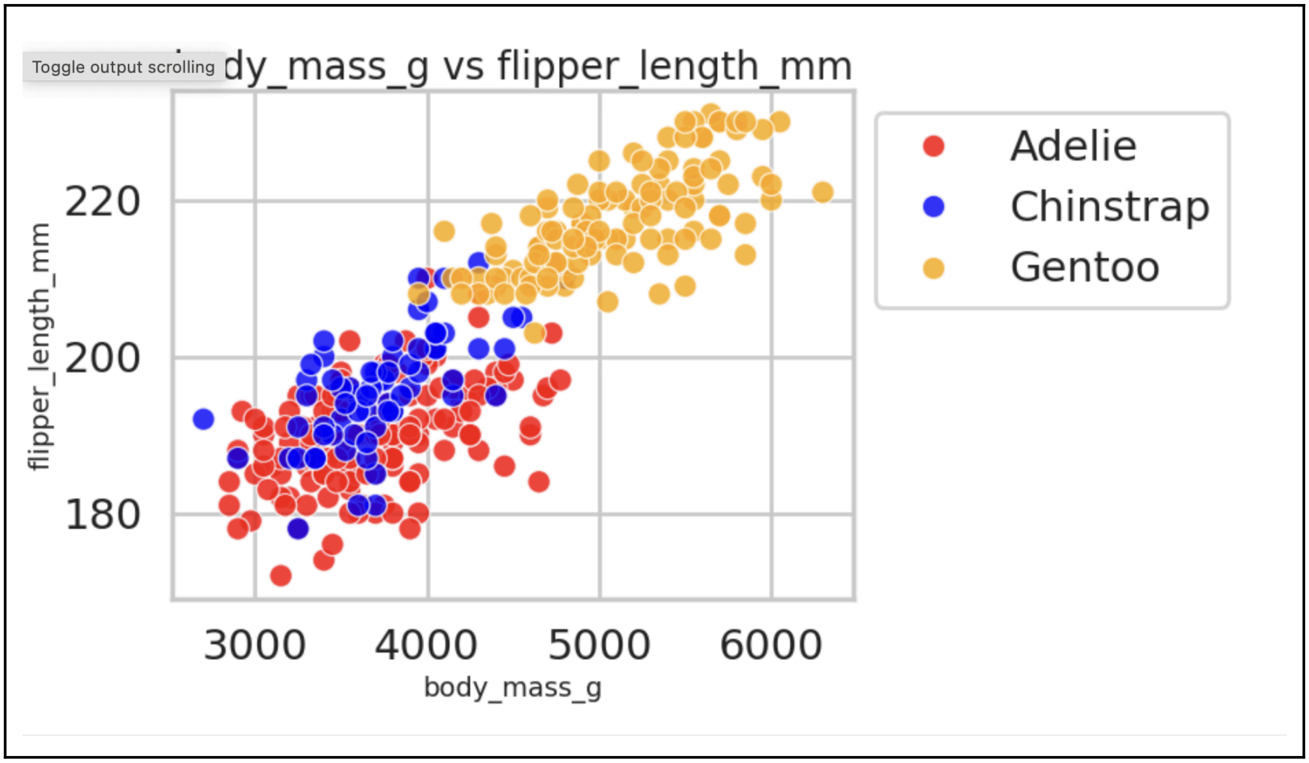
Training **Error** (%)	Validation **Error** (%)	Test **Error** (%)
0.41%	2.46%	2.47%

3. Species Prediction

a. Visualization of Features

Include at least two scatterplots of pairs of features.

Visualization (labels should make clear which features are used)




You may extend the table if you have more results

Of these three options, which two features (by themselves) are best able to classify the penguin species?

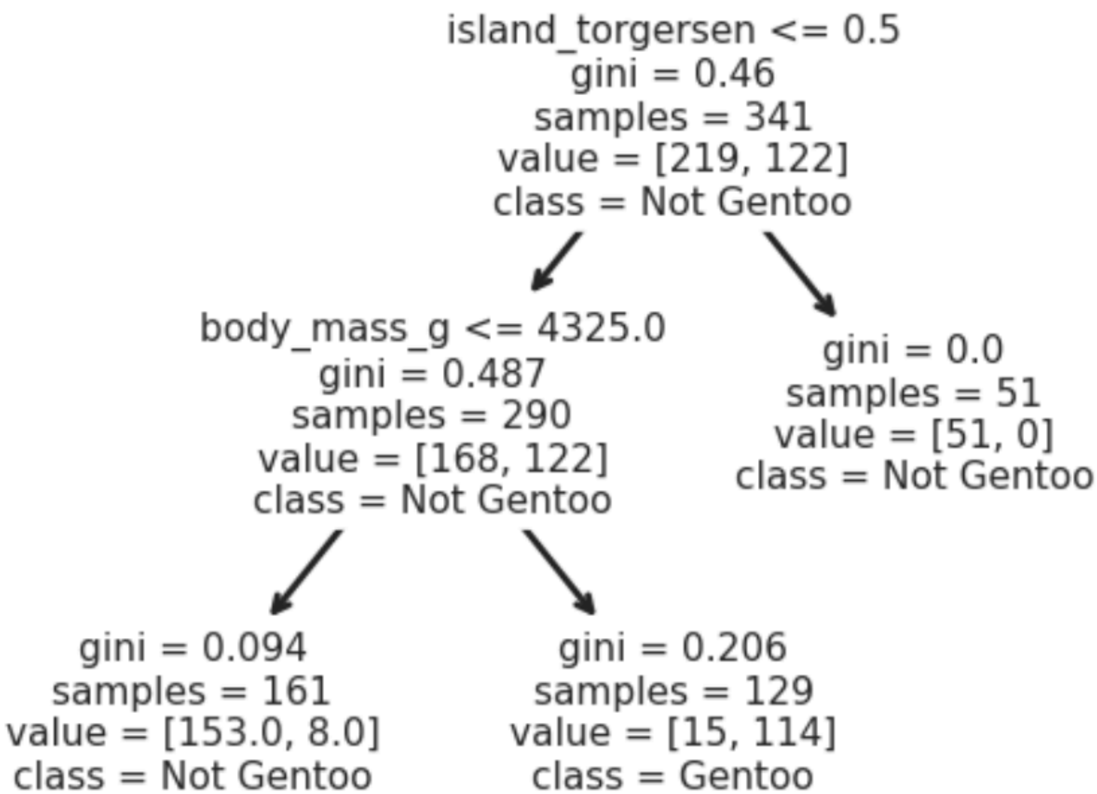
1. Culmen Depth + Flipper Length
2. Flipper Length + Culmen Length
3. Flipper Length + Body Mass

2. Flipper Length + Culmen Length

b. Simple rule to identify Gentoo

Display your decision tree with labeled features and classes.

```
|--- island_torgersen <= 0.50
|   |--- body_mass_g <= 4325.00
|   |   |--- class: Not Gentoo
|   |--- body_mass_g >  4325.00
|   |   |--- class: Gentoo
|--- island_torgersen >  0.50
|   |--- class: Not Gentoo
```



Write down the simple two-part rule to identify Gentoo. For example, the format should be "If Mass > 3000 and Culmen Depth < 17, then species is Gentoo".

If…

island_torgersen <= 0.50

and

body_mass_g > 4325.00

then species is Gentoo.

Rule precision: fraction of penguins that satisfy this rule that are Gentoos (# gentoo predicted / # predicted)

114 / 129

Rule recall: fraction of all Gentoo penguins that are identified as Gentoo using this rule (# gentoo predicted / # gentoo)

114 / 122

c. Model Design

Describe the model that achieves best 5-fold cross-validation accuracy:

GradientBoostingClassifier with default parameters

5-fold Cross-Validation Accuracy: (xx.x%)

98.8

3. Stretch Goals

a. Improve MNIST Classification Performance using MLPs

Report the classification val and test errors and details of your best method. Describe your approach and parameters. Feel free to change the MLP batch size, optimizer (e.g. try Adam), learning rate, number of epochs, hidden layer size, activation layer, or anything else.

Description and key parameters

Optimizer = Adam Hidden layer(s) = 2 layers (512, 256) Learning rate = 0.05 Number of epochs = 26 Any other details: - Features normalized using mean and std - Manually stopped training

Validation **Error** (%)	Test **Error** (%)
1.61%	1.72%

b. Find a second simple rule to identify Gentoo

Provide the second two-part rule here (that is substantially different from your first rule).

If…

and

then species is Gentoo.

Rule precision: fraction of penguins that satisfy this rule that are Gentoos (# gentoo predicted / # predicted)

Rule recall: fraction of all Gentoo penguins that are identified as Gentoo using this rule (# gentoo predicted / # gentoo)

c. Positional encoding

Show the RGB image obtained by predicting directly from (x,y) and the image obtained by predicting from the positional encoding.

Input to network is (x,y)

Input to network is pos_enc(x, y)

Acknowledgments / Attribution

List any outside sources for code or improvement ideas or “None”.

CS441_SP24_HW4_Starter

April 4, 2024

0.1 CS441: Applied ML - HW 4

0.1.1 Part 1: Model Complexity and Tree-based Regressors

One measure of a tree's complexity is the maximum tree depth. Train tree, random forest, and boosted tree regressors on the temperature regression task, using all default parameters except:

- max_depth={2,4,8,16,32}
- random_state=0
- For random forest: max_features=1/3

Measure train and val RMSE for each and plot them all on the same plot using the provided plot_depth_error function. You should have six lines (train/val for each model type), each with 5 data points (one for each max depth value). Include the plot and answer the analysis questions in the report.

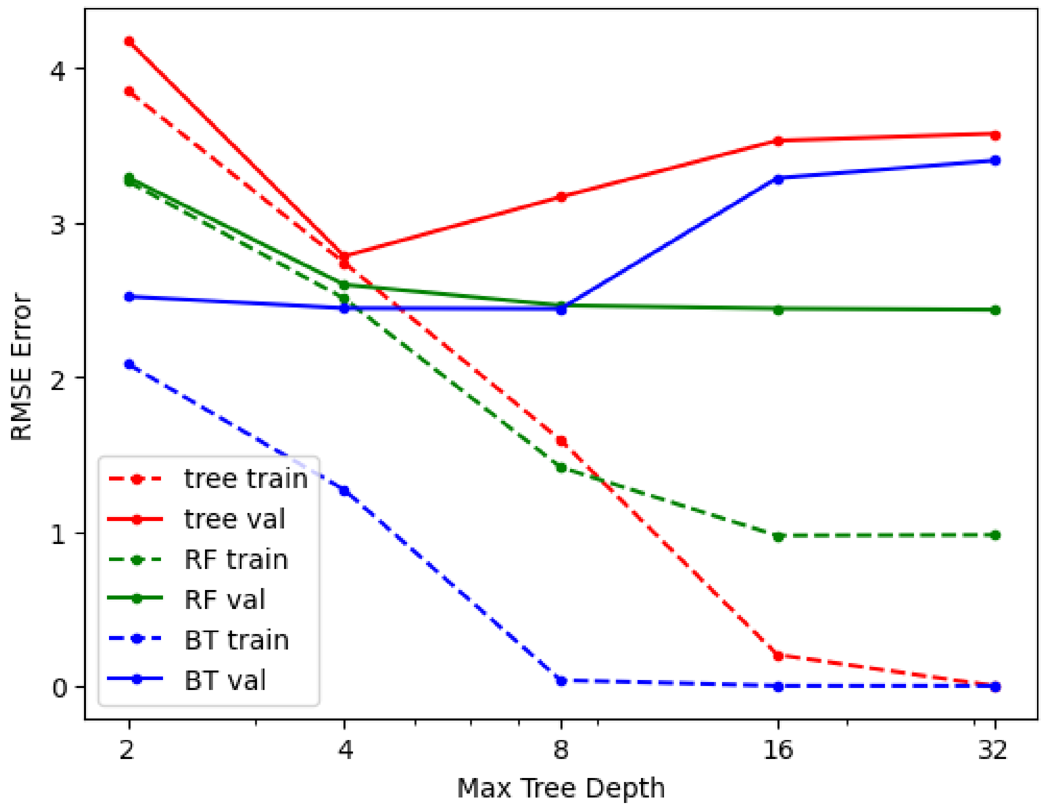
[1]:
```
import numpy as np
# from google.colab import drive
%matplotlib inline
from matplotlib import pyplot as plt

# load data (modify to match your data directory or comment)
def load_temp_data():
  # drive.mount('/content/drive')
  # datadir = "/content/drive/My Drive/CS441/24SP/hw1/"
  datadir="./"
  T = np.load(datadir + 'temperature_data.npz')
  x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val,␣
↪dates_test, feature_to_city, feature_to_day = \
  T['x_train'], T['y_train'], T['x_val'], T['y_val'], T['x_test'], T['y_test'],␣
↪T['dates_train'], T['dates_val'], T['dates_test'], T['feature_to_city'],␣
↪T['feature_to_day']
  return (x_train, y_train, x_val, y_val, x_test, y_test, dates_train,␣
↪dates_val, dates_test, feature_to_city, feature_to_day)

# plot one data point for listed cities and target temperature
def plot_temps(x, y, cities, feature_to_city, feature_to_day, target_date):
  nc = len(cities)
  ndays = 5
  xplot = np.array([-5,-4,-3,-2,-1])
```

```
    yplot = np.zeros((nc,ndays))
    for f in np.arange(len(x)):
      for c in np.arange(nc):
        if cities[c]==feature_to_city[f]:
          yplot[feature_to_day[f]+ndays,c] = x[f]
    plt.plot(xplot,yplot)
    plt.legend(cities)
    plt.plot(0, y, 'b*', markersize=10)
    plt.title('Predict Temp for Cleveland on ' + target_date)
    plt.xlabel('Day')
    plt.ylabel('Avg Temp (C)')
    plt.show()

# load data
(x_train, y_train, x_val, y_val, x_test, y_test, dates_train, dates_val,␣
 ↪dates_test, feature_to_city, feature_to_day) = load_temp_data()
```

[2]:
```
# to plot the errors
def plot_depth_error(max_depths, tree_train_err, tree_val_err, rf_train_err,␣
 ↪rf_val_err, bt_train_err, bt_val_err):
    plt.figure()
    plt.semilogx(max_depths, tree_train_err, 'r.--',label='tree train')
    plt.semilogx(max_depths, tree_val_err, 'r.-', label='tree val')
    plt.semilogx(max_depths, rf_train_err, 'g.--',label='RF train')
    plt.semilogx(max_depths, rf_val_err, 'g.-', label='RF val')
    plt.semilogx(max_depths, bt_train_err, 'b.--',label='BT train')
    plt.semilogx(max_depths, bt_val_err, 'b.-', label='BT val')
    plt.ylabel('RMSE Error')
    plt.xlabel('Max Tree Depth')
    plt.xticks(max_depths, max_depths)
    plt.legend()
    plt.rcParams.update({'font.size': 20})
    plt.show()
```

[4]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor

max_depths = [2,4,8,16,32]

# usage examples
# model = DecisionTreeRegressor(random_state=0, max_depth=max_depth)
# model = RandomForestRegressor(random_state=0, max_depth=max_depth,␣
 ↪max_features=1/3)
# model = GradientBoostingRegressor(random_state=0, max_depth=max_depth)
```

[16]:
```
def rmse(a,b):
  return np.sqrt(((a - b) ** 2).mean())

tree_train_err, tree_val_err, rf_train_err, rf_val_err, bt_train_err,␣
↪bt_val_err = [[] for i in range(6)]

for max_depth in max_depths:
  model = DecisionTreeRegressor(random_state=0,max_depth=max_depth)
  model.fit(x_train,y_train)
  tree_train_err.append(rmse(y_train,model.predict(x_train)))
  tree_val_err.append(rmse(y_val,model.predict(x_val)))

  model = RandomForestRegressor(random_state=0, max_depth=max_depth,␣
↪max_features=1/3)
  model.fit(x_train,y_train)
  rf_train_err.append(rmse(y_train,model.predict(x_train)))
  rf_val_err.append(rmse(y_val,model.predict(x_val)))

  model = GradientBoostingRegressor(random_state=0, max_depth=max_depth)
  model.fit(x_train,y_train)
  bt_train_err.append(rmse(y_train,model.predict(x_train)))
  bt_val_err.append(rmse(y_val,model.predict(x_val)))


plot_depth_error(max_depths,tree_train_err, tree_val_err, rf_train_err,␣
↪rf_val_err, bt_train_err, bt_val_err)
```



0.1.2 Part 2: MLPs with MNIST

For this part, you will want to use a GPU to improve runtime. Google Colab provides limited free GPU acceleration to all users. Go to Runtime and change Runtime Type to GPU. This will reset your compute node, so do it before starting to run other cells.

See Tips for detailed guidance on this problem.

First, use PyTorch to implement a Multilayer Perceptron network with one hidden layer (size 64) with ReLU activation. Set the network to minimize cross-entropy loss, which is the negative log probability of the training labels given the training features. This objective function takes unnormalized logits as inputs.

Do not use MLP in sklearn for this HW - use Torch.

[1]:
```
# initialization code
import numpy as np
from keras.datasets import mnist
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats
import torch
import torch.nn as nn
```

```python
def load_mnist():
  '''
  Loads, reshapes, and normalizes the data
  '''
  (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST data
  x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d␣
↪vectors
  x_test = np.reshape(x_test, (len(x_test), 28*28))
  maxval = x_train.max()
  x_train = x_train/maxval  # normalize values to range from 0 to 1
  x_test = x_test/maxval
  return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
  '''
  Displays one or more examples in a row or a grid
  '''
  if subplot_rows>1 or subplot_cols>1:
    fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
    for i in np.arange(len(x)):
      ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
      ax[i].axis('off')
  else:
      plt.imshow(np.reshape(x, (28,28)), cmap='gray')
      plt.axis('off')
  plt.show()
```

```
2024-04-04 22:28:11.317892: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-04-04 22:28:11.317982: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-04-04 22:28:11.617690: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-04-04 22:28:12.287867: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2024-04-04 22:28:16.969390: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
```

```
find TensorRT
```

[2]:
```
# Sets device to "cuda" if a GPU is available  (in Colabs, enable GPU by␣
 ↪Edit->Notebook Settings-->Hardware Accelerator=GPU)
device = "cuda" if torch.cuda.is_available() else 'cpu'
print(device) # make sure you're using GPU instance
```

```
cuda
```

2a Using the train/val split provided in the starter code, train your network for 100 epochs with learning rates of 0.01, 0.1, and 1. Use a batch size of 256 and the SGD optimizer. After each epoch, record the mean training and validation loss and compute the validation error of the final model. The mean validation loss should be computed after the epoch is complete. The mean training loss can either be computed after the epoch is complete, or, for efficiency, computed using the losses accumulated during the training of the epoch. Plot the training and validation losses using the display_error_curves function.

[3]:
```
(x_train, y_train), (x_test, y_test) = load_mnist()

# create train/val split
ntrain = 50000
x_val = x_train[ntrain:].copy()
y_val = y_train[ntrain:].copy()
x_train = x_train[:ntrain]
y_train = y_train[:ntrain]
x_val.shape
```

[3]:
```
(10000, 784)
```

[4]:
```
def display_error_curves(training_losses, validation_losses):
  """
  Plots the training and validation loss curves
  training_losses and validation_losses should be lists or arrays of the same␣
 ↪length
  """
  num_epochs = len(training_losses)

  plt.plot(range(num_epochs), training_losses, label="Training Loss")
  plt.plot(range(num_epochs), validation_losses, label="Validation Loss")

  # Add in a title and axes labels
  plt.title('Training and Validation Loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')

  # Display the plot
  plt.legend(loc='best')
```

```python
plt.show()
```

[5]:
```python
# Define the model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        self.l1 = nn.Linear(input_size,hidden_size)
        self.l2 = nn.Linear(hidden_size,output_size)
        self.act = nn.ReLU()

    def forward(self, x):
        return self.l2(self.act(self.l1(x)))
```

[6]:
```python
# This is a possible function definition for training MLP, but feel free to␣
↪change it
# You may also want to create helper functions, e.g. for computing loss or␣
↪prediction
def train_MLP_mnist(train_loader, val_loader, lr=1e-1, num_epochs=100):
  '''
  Train a MLP
  Input: train_loader and val_loader are dataloaders for the training and
  val data, respectively. lr is the learning rate, and the network will
  be trained for num_epochs epochs.
  Output: return a trained MLP
  '''
  # TODO: fill in all code

  input_size = 784
  hidden_size = 64
  output_size = 10

  # Instantiate the model
  mlp = MLP(input_size,hidden_size,output_size)
  mlp.to(device)

  # Train the model, compute and store train/val loss at each epoch
  train_losses,val_losses = [],[]
  optim = torch.optim.SGD(mlp.parameters(),lr=lr)
  for e in range(num_epochs):
    print(f"Epoch: {e}")
    tloss, terr = evaluate_MLP(mlp,train_loader,optim)
    vloss, verr = evaluate_MLP(mlp,val_loader,None)
    train_losses.append(tloss)
    val_losses.append(vloss)
    print(f"Terr {terr}   verr {verr}")
    print(f"Tloss {tloss} vloss {vloss}")
```

```
  # Display Loss Curves
  display_error_curves(train_losses,val_losses)
  print(np.min(val_losses), np.argmin(val_losses))
  return mlp


def evaluate_MLP(mlp, loader, optim):
  ''' Computes loss and error rate given your mlp model and data loader'''
  N = 0
  acc = 0
  loss = 0
  loss_function = torch.nn.CrossEntropyLoss()
  with torch.set_grad_enabled(optim is not None):
    for i, data in enumerate(loader, 0):
      # print(i)
      # Get inputs
      inputs, targets = data
      N += len(targets)

      # Perform forward pass
      outputs = mlp(inputs.to(device))

      # Compute sum of correct labels
      y_pred = np.argmax(outputs.detach().cpu().numpy(), axis=1)
      y_gt = np.argmax(targets.numpy(), axis=1)
      acc += np.sum(y_pred==y_gt)

      # Compute loss
      L = loss_function(outputs, targets.to(device))
      if optim:
        optim.zero_grad()
        L.backward()
        optim.step()
      loss += L.item()*len(targets)


  loss /= N
  acc /= N

  return loss, 1-acc
```

[7]:
```
# Code for running experiments

print(device) # make sure you're using GPU instance
torch.manual_seed(0) # to avoid randomness, but if you wanted to create an␣
 ↪ensemble, you should not use a manual seed
```

```
# TODO (set up dataloaders, and call training function)
trainset = torch.utils.data.TensorDataset(torch.Tensor(x_train), torch.
 ↪Tensor(np.eye(10)[y_train]))
train_loader = torch.utils.data.DataLoader(trainset, batch_size=256,␣
 ↪shuffle=True, num_workers=0)

valset = torch.utils.data.TensorDataset(torch.Tensor(x_val), torch.Tensor(np.
 ↪eye(10)[y_val]))
val_loader = torch.utils.data.DataLoader(valset, batch_size=1000, shuffle=True,␣
 ↪num_workers=0)
```

```
cuda
```

[8]:
```
testset = torch.utils.data.TensorDataset(torch.Tensor(x_test), torch.Tensor(np.
 ↪eye(10)[y_test]))
test_loader = torch.utils.data.DataLoader(testset, batch_size=1000,␣
 ↪shuffle=True, num_workers=0)
```

[9]:
```
torch.manual_seed(0)
mlp = train_MLP_mnist(train_loader,val_loader,.1)
```

```
Epoch: 0
Terr 0.20704   verr 0.11240000000000006
Tloss 0.8635518909454346 vloss 0.4086389631032944
Epoch: 1
Terr 0.10518000000000005   verr 0.09040000000000004
Tloss 0.3782551777935028 vloss 0.3238971889019012
Epoch: 2
Terr 0.09284000000000003   verr 0.08650000000000002
Tloss 0.32524825715065003 vloss 0.2945525795221329
Epoch: 3
Terr 0.08440000000000003   verr 0.0746
Tloss 0.2957117403411865 vloss 0.26729006320238113
Epoch: 4
Terr 0.07718000000000003   verr 0.06840000000000002
Tloss 0.2725549217414856 vloss 0.243122835457325
Epoch: 5
Terr 0.07155999999999996   verr 0.06699999999999995
Tloss 0.25301938406944274 vloss 0.23046422004699707
Epoch: 6
Terr 0.06645999999999996   verr 0.06030000000000002
Tloss 0.23652801489830017 vloss 0.21672380119562148
Epoch: 7
Terr 0.06240000000000001   verr 0.05640000000000006
Tloss 0.22202131447076798 vloss 0.2057206466794014
Epoch: 8
```

```
Terr 0.05891999999999997   verr 0.05579999999999996
Tloss 0.20898935275554656 vloss 0.1961863398551941
Epoch: 9
Terr 0.05618000000000001   verr 0.053300000000000014
Tloss 0.19767102230072023 vloss 0.19073049277067183
Epoch: 10
Terr 0.05335999999999996   verr 0.04959999999999998
Tloss 0.1875739734172821 vloss 0.17991431057453156
Epoch: 11
Terr 0.051080000000000014   verr 0.0474
Tloss 0.17841224222183227 vloss 0.17215566635131835
Epoch: 12
Terr 0.04834000000000005   verr 0.04510000000000003
Tloss 0.16997942166805266 vloss 0.1659148022532463
Epoch: 13
Terr 0.04603999999999997   verr 0.04410000000000003
Tloss 0.16223600018024445 vloss 0.16089427024126052
Epoch: 14
Terr 0.044340000000000046   verr 0.04290000000000005
Tloss 0.15525528495550156 vloss 0.15486137866973876
Epoch: 15
Terr 0.04261999999999999   verr 0.04239999999999999
Tloss 0.1487938488149643 vloss 0.1535654902458191
Epoch: 16
Terr 0.041019999999999945   verr 0.044399999999999995
Tloss 0.1429458200740814 vloss 0.15757024437189102
Epoch: 17
Terr 0.03990000000000005   verr 0.040200000000000014
Tloss 0.13767276611328125 vloss 0.14253877848386765
Epoch: 18
Terr 0.038020000000000054   verr 0.0403
Tloss 0.13227475219011306 vloss 0.14192736893892288
Epoch: 19
Terr 0.036619999999999986   verr 0.03859999999999997
Tloss 0.1277863086748123 vloss 0.13681117594242095
Epoch: 20
Terr 0.03503999999999996   verr 0.03700000000000003
Tloss 0.12326760881900788 vloss 0.13206204175949096
Epoch: 21
Terr 0.034540000000000015   verr 0.03720000000000001
Tloss 0.11906295682907105 vloss 0.1326569102704525
Epoch: 22
Terr 0.03264   verr 0.03720000000000001
Tloss 0.1151627186524868 vloss 0.12670560628175737
Epoch: 23
Terr 0.03166000000000002   verr 0.0363
Tloss 0.11159292246818542 vloss 0.12442973777651786
Epoch: 24
```

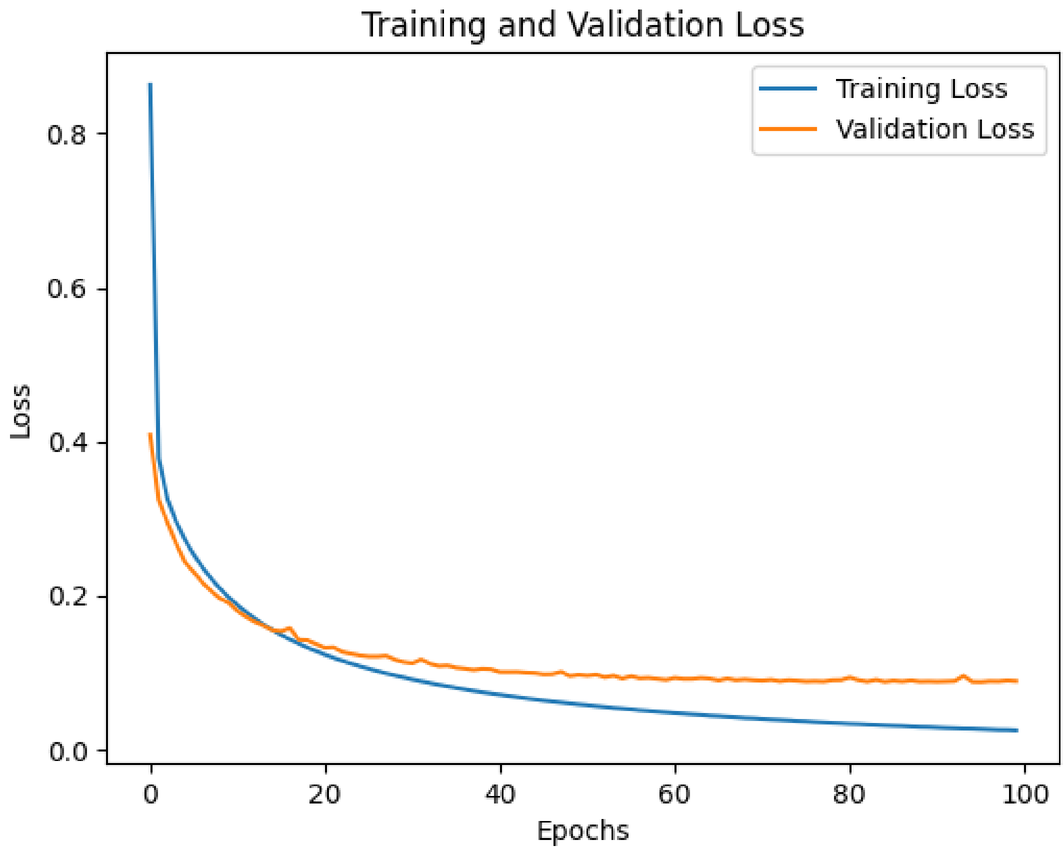
```
Terr 0.030059999999999976   verr 0.03310000000000002
Tloss 0.1080906416463852 vloss 0.12220851331949234
Epoch: 25
Terr 0.02959999999999996   verr 0.03320000000000001
Tloss 0.10492640667676925 vloss 0.12085393145680427
Epoch: 26
Terr 0.028220000000000023   verr 0.033499999999999974
Tloss 0.10179907207250595 vloss 0.12082005813717842
Epoch: 27
Terr 0.026920000000000055   verr 0.033399999999999985
Tloss 0.09881959172964096 vloss 0.12171831727027893
Epoch: 28
Terr 0.026680000000000037   verr 0.03180000000000005
Tloss 0.09619653372049332 vloss 0.11668747961521149
Epoch: 29
Terr 0.025560000000000027   verr 0.031399999999999983
Tloss 0.09357097613215447 vloss 0.11354890167713165
Epoch: 30
Terr 0.02507999999999999   verr 0.03159999999999996
Tloss 0.09091779416561127 vloss 0.11209376603364944
Epoch: 31
Terr 0.02432000000000001   verr 0.03359999999999996
Tloss 0.08868565662860871 vloss 0.11689029559493065
Epoch: 32
Terr 0.02414000000000005   verr 0.03149999999999997
Tloss 0.0864179239320755 vloss 0.11152919828891754
Epoch: 33
Terr 0.02300000000000002   verr 0.029900000000000038
Tloss 0.0840595360136032 vloss 0.10844598561525345
Epoch: 34
Terr 0.022379999999999955   verr 0.028699999999999948
Tloss 0.08208345973968506 vloss 0.1090236559510231
Epoch: 35
Terr 0.02188000000000001   verr 0.029299999999999993
Tloss 0.0800791757684946 vloss 0.10625712275505066
Epoch: 36
Terr 0.021279999999999966   verr 0.02959999999999996
Tloss 0.07814149736702442 vloss 0.10493572428822517
Epoch: 37
Terr 0.021039999999999948   verr 0.028699999999999948
Tloss 0.07617610409975052 vloss 0.103452268242836
Epoch: 38
Terr 0.020519999999999983   verr 0.02839999999999998
Tloss 0.07440524766921996 vloss 0.10483208447694778
Epoch: 39
Terr 0.020199999999999996   verr 0.029900000000000038
Tloss 0.07271677977323532 vloss 0.10421483218669891
Epoch: 40
```

```
Terr 0.01959999999999995   verr 0.027900000000000036
Tloss 0.07117000739812851 vloss 0.10061295107007026
Epoch: 41
Terr 0.019379999999999953   verr 0.029000000000000026
Tloss 0.06959554381370545 vloss 0.1005745254456997
Epoch: 42
Terr 0.018419999999999992   verr 0.028100000000000014
Tloss 0.06804282830834389 vloss 0.10063475221395493
Epoch: 43
Terr 0.018279999999999963   verr 0.027800000000000047
Tloss 0.0664820098245144 vloss 0.09970689862966538
Epoch: 44
Terr 0.017819999999999947   verr 0.027900000000000036
Tloss 0.06508918017268181 vloss 0.09921436458826065
Epoch: 45
Terr 0.01739999999999997   verr 0.027100000000000013
Tloss 0.06363911603450775 vloss 0.09756950661540031
Epoch: 46
Terr 0.016900000000000026   verr 0.027000000000000024
Tloss 0.06235984413504601 vloss 0.09781367406249046
Epoch: 47
Terr 0.016680000000000028   verr 0.02959999999999996
Tloss 0.06109026929616928 vloss 0.10084009617567062
Epoch: 48
Terr 0.016040000000000054   verr 0.028100000000000014
Tloss 0.0598524529337883 vloss 0.09549872502684593
Epoch: 49
Terr 0.015800000000000036   verr 0.028299999999999992
Tloss 0.05863088870048523 vloss 0.09709331318736077
Epoch: 50
Terr 0.015179999999999971   verr 0.028100000000000014
Tloss 0.05756268373012543 vloss 0.09598132446408272
Epoch: 51
Terr 0.014900000000000024   verr 0.028900000000000037
Tloss 0.05626062457084656 vloss 0.0973051831126213
Epoch: 52
Terr 0.014580000000000037   verr 0.026699999999999946
Tloss 0.05519695436477661 vloss 0.0939082458615303
Epoch: 53
Terr 0.01419999999999999   verr 0.027000000000000024
Tloss 0.053955083644390105 vloss 0.09589515700936317
Epoch: 54
Terr 0.013959999999999972   verr 0.026900000000000035
Tloss 0.05306482212483883 vloss 0.09209161251783371
Epoch: 55
Terr 0.013460000000000027   verr 0.026699999999999946
Tloss 0.05214876147985458 vloss 0.09518095180392265
Epoch: 56
```

```
Terr 0.012979999999999992   verr 0.027599999999999958
Tloss 0.05096778401017189 vloss 0.09253189265727997
Epoch: 57
Terr 0.01312000000000002   verr 0.027800000000000047
Tloss 0.050155787482261655 vloss 0.09307094514369965
Epoch: 58
Terr 0.013040000000000052   verr 0.027100000000000013
Tloss 0.049161160529851915 vloss 0.09176949188113212
Epoch: 59
Terr 0.012340000000000018   verr 0.0262
Tloss 0.04826654378890991 vloss 0.09050279781222344
Epoch: 60
Terr 0.012299999999999978   verr 0.025900000000000034
Tloss 0.04732055198788643 vloss 0.09301351904869079
Epoch: 61
Terr 0.01200000000000001   verr 0.025900000000000034
Tloss 0.046569135185480115 vloss 0.09169436469674111
Epoch: 62
Terr 0.011759999999999993   verr 0.028000000000000025
Tloss 0.04566480160057545 vloss 0.09176656901836396
Epoch: 63
Terr 0.011440000000000006   verr 0.02639999999999998
Tloss 0.04497075219631195 vloss 0.09296715520322323
Epoch: 64
Terr 0.01100000000000001   verr 0.02749999999999997
Tloss 0.04396974532365799 vloss 0.0921167328953743
Epoch: 65
Terr 0.01085999999999998   verr 0.025499999999999967
Tloss 0.04336730348348618 vloss 0.08972727470099925
Epoch: 66
Terr 0.010639999999999983   verr 0.0262
Tloss 0.042580240699052814 vloss 0.09216598719358444
Epoch: 67
Terr 0.010700000000000043   verr 0.027000000000000024
Tloss 0.041875487059056755 vloss 0.09006521180272102
Epoch: 68
Terr 0.010000000000000009   verr 0.02529999999999999
Tloss 0.04094658839941025 vloss 0.09107502773404122
Epoch: 69
Terr 0.009979999999999989   verr 0.026699999999999946
Tloss 0.040510121207237244 vloss 0.0900398164987564
Epoch: 70
Terr 0.009680000000000022   verr 0.026599999999999957
Tloss 0.03969102276802063 vloss 0.08947280943393707
Epoch: 71
Terr 0.009419999999999984   verr 0.027000000000000024
Tloss 0.0390293904709816 vloss 0.09052354395389557
Epoch: 72
```

```
Terr 0.009440000000000004   verr 0.02510000000000001
Tloss 0.038494434577226636 vloss 0.08873282410204411
Epoch: 73
Terr 0.008979999999999988   verr 0.026100000000000012
Tloss 0.03787808868944645 vloss 0.09007946476340294
Epoch: 74
Terr 0.008939999999999948   verr 0.025800000000000045
Tloss 0.03703799706816673 vloss 0.08936624974012375
Epoch: 75
Terr 0.008360000000000034   verr 0.026000000000000023
Tloss 0.036567655780911444 vloss 0.08843703903257846
Epoch: 76
Terr 0.008299999999999974   verr 0.0262
Tloss 0.03574925950378179 vloss 0.08877413347363472
Epoch: 77
Terr 0.008079999999999976   verr 0.02529999999999999
Tloss 0.035185120783448216 vloss 0.08839048817753792
Epoch: 78
Terr 0.007920000000000038   verr 0.02529999999999999
Tloss 0.03453790689945221 vloss 0.09001660160720348
Epoch: 79
Terr 0.007719999999999949   verr 0.025800000000000045
Tloss 0.0340900598436594 vloss 0.09024174809455872
Epoch: 80
Terr 0.00744000000000002   verr 0.026900000000000035
Tloss 0.03343914961338043 vloss 0.09349655956029893
Epoch: 81
Terr 0.007340000000000013   verr 0.02639999999999998
Tloss 0.03310830275893211 vloss 0.09014232978224754
Epoch: 82
Terr 0.007480000000000042   verr 0.025699999999999945
Tloss 0.032516841430664065 vloss 0.08828605785965919
Epoch: 83
Terr 0.007140000000000035   verr 0.025399999999999978
Tloss 0.032002154606580735 vloss 0.09052772149443626
Epoch: 84
Terr 0.006780000000000008   verr 0.024800000000000044
Tloss 0.03142536310613155 vloss 0.08805048763751984
Epoch: 85
Terr 0.006659999999999999   verr 0.025499999999999967
Tloss 0.03096854523807764 vloss 0.0894688293337822
Epoch: 86
Terr 0.006639999999999979   verr 0.02629999999999999
Tloss 0.030614767703413964 vloss 0.0884522657841444
Epoch: 87
Terr 0.006120000000000014   verr 0.025399999999999978
Tloss 0.029947758051753044 vloss 0.08979427218437194
Epoch: 88
```

```
Terr 0.005939999999999945   verr 0.025399999999999978
Tloss 0.029360919052362443 vloss 0.0883260142058134
Epoch: 89
Terr 0.005979999999999985   verr 0.025000000000000022
Tloss 0.029165867356657982 vloss 0.08855071812868118
Epoch: 90
Terr 0.005739999999999967   verr 0.025699999999999945
Tloss 0.02857176577985287 vloss 0.08822127655148507
Epoch: 91
Terr 0.005519999999999969   verr 0.0252
Tloss 0.028045215773582457 vloss 0.08855902850627899
Epoch: 92
Terr 0.00544   verr 0.025499999999999967
Tloss 0.027692427901029586 vloss 0.08897810354828835
Epoch: 93
Terr 0.00548000000000004   verr 0.02629999999999999
Tloss 0.02725810732603073 vloss 0.09570439383387566
Epoch: 94
Terr 0.005560000000000009   verr 0.024599999999999955
Tloss 0.027042798011302948 vloss 0.08797539919614791
Epoch: 95
Terr 0.005179999999999962   verr 0.02529999999999999
Tloss 0.026553344784677028 vloss 0.08776346370577812
Epoch: 96
Terr 0.005260000000000042   verr 0.02510000000000001
Tloss 0.02608896798580885 vloss 0.08875067457556725
Epoch: 97
Terr 0.0047399999999999665   verr 0.026100000000000012
Tloss 0.025545992239415647 vloss 0.08860600516200065
Epoch: 98
Terr 0.0049599999999999644   verr 0.025000000000000022
Tloss 0.02543264357686043 vloss 0.08962047286331654
Epoch: 99
Terr 0.0046399999999999775   verr 0.024599999999999955
Tloss 0.02494989398300648 vloss 0.08898630291223526
```




```
0.08776346370577812 95
```

2b Based on the loss curves, select the learning rate and number of epochs that minimizes the validation loss. Retrain that model (if it's not stored), and report training loss, validation loss, training error, validation error, and test error.

[12]:
```
N = 0
acc = 0
loss = 0
loss_function = torch.nn.CrossEntropyLoss()
with torch.set_grad_enabled(False):
  for i, data in enumerate(train_loader, 0):
    # print(i)
    # Get inputs
    inputs, targets = data
    N += len(targets)

    # Perform forward pass
    outputs = mlp(inputs.to(device))
```

```
    # Compute sum of correct labels
    y_pred = np.argmax(outputs.detach().cpu().numpy(), axis=1)
    y_gt = np.argmax(targets.numpy(), axis=1)
    acc += np.sum(y_pred==y_gt)

    # Compute loss
    L = loss_function(outputs, targets.to(device))
    loss += L.item()*len(targets)


loss /= N
acc /= N

print(loss,1-acc)
```

```
0.02350550983905792 0.004079999999999725
```

0.2 Part 3: Predicting Penguin Species

Include all your code for part 3 in this section.

[1]:
```
import numpy as np
# from google.colab import drive
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
#styling preferences for sns
sns.set_style('whitegrid')
sns.set_context('poster')
# drive.mount('/content/gdrive/')
# datadir = "/content/gdrive/MyDrive/CS441/hw4/" # TO DO: modify this to your␣
 ↪directory
datadir="./"
df_penguins = pd.read_csv(datadir + 'penguins_size.csv')
df_penguins.head(10)

# convert features with multiple string values to binary features so they can␣
 ↪be used by sklearn
def get_penguin_xy(df_penguins):
  data = np.array(df_penguins[['island', 'culmen_length_mm', 'culmen_depth_mm',␣
 ↪'flipper_length_mm', 'body_mass_g', 'sex']])
  y = df_penguins['species']
  ui = np.unique(data[:,0]) # unique island
  us = np.unique(data[:,-1]) # unique sex
  X = np.zeros((len(y), 10))
  for i in range(len(y)):
```

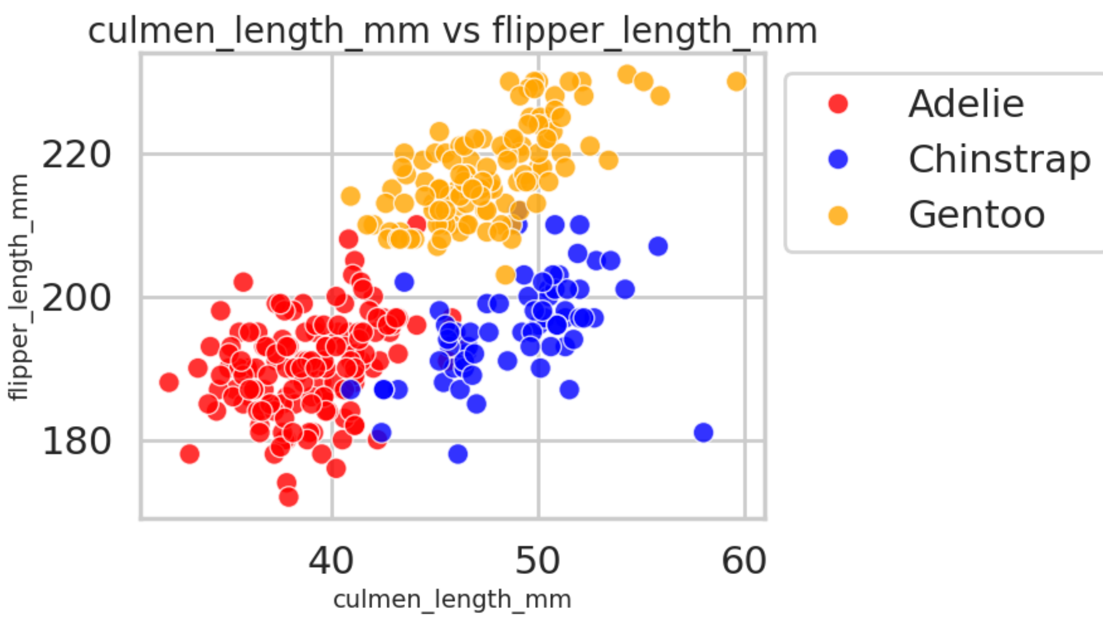
```
    f = 0
    for j in range(len(ui)):
      if data[i, f]==ui[j]:
        X[i, f+j] = 1
    f = f + len(ui)
    X[i, f:(f+4)] = data[i, 1:5]
    f=f+4
    for j in range(len(us)):
      if data[i, 5]==us[j]:
        X[i, f+j] = 1
  feature_names = ['island_biscoe', 'island_dream', 'island_torgersen',␣
↪'culmen_length_mm', 'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g',␣
↪'sex_female', 'sex_male', 'sex_unknown']
  X = pd.DataFrame(X, columns=feature_names)
  return(X, y, feature_names, np.unique(y))
```

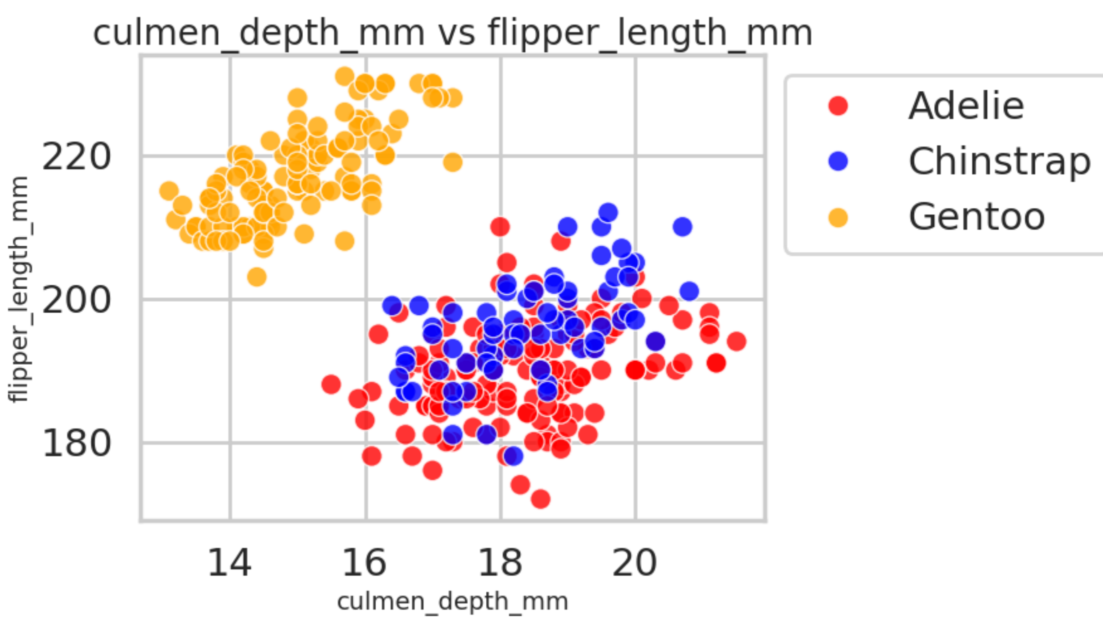
```
/tmp/ipykernel_320/2524931758.py:5: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of
pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better
interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

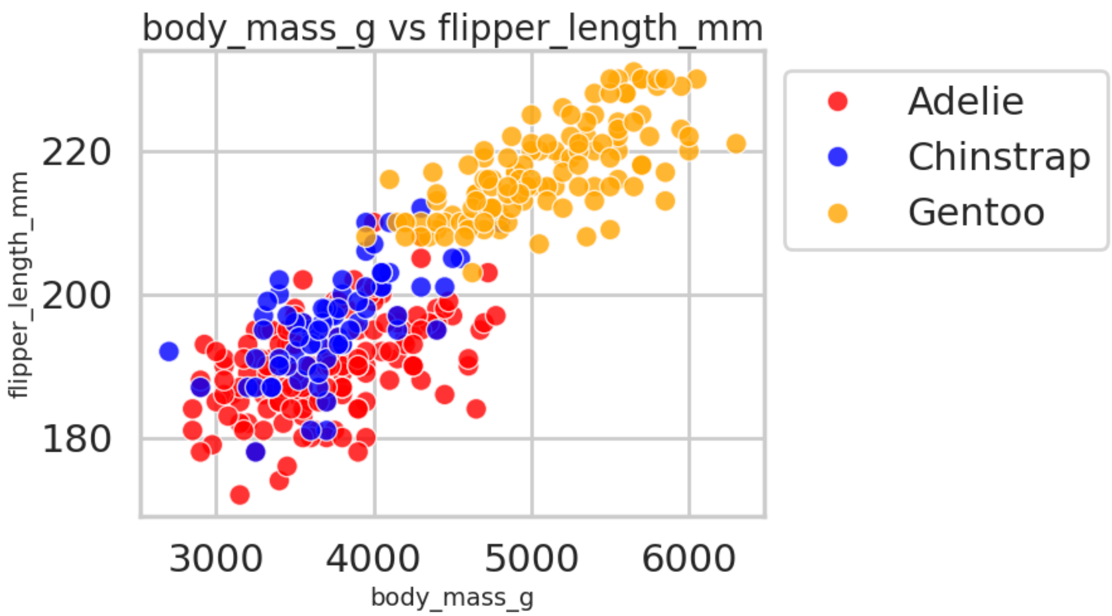
3a Spend some time to visualize different pairs of features and their relationships to the species. We've done one for you. Include in your report at least two other visualizations.

[2]:
```
def plot_scatter(feature1, feature2):
  '''
  Provide names of two features to create a scatterplot of them
  E.g. plot_scatter('culmen_length_mm', 'culmen_depth_mm')
  Possible features: 'culmen_length_mm', 'culmen_depth_mm',␣
↪'flipper_length_mm', 'body_mass_g'
  '''

  palette = ["red", "blue", "orange"]

  sns.scatterplot(data=df_penguins, x = feature1, y = feature2,
               hue = 'species', palette=palette, alpha=0.8)
  # Doc: https://seaborn.pydata.org/generated/seaborn.scatterplot.html

  plt.xlabel(feature1, fontsize=14)
  plt.ylabel(feature2, fontsize=14)
  plt.title(feature1 + ' vs ' + feature2, fontsize=20)
```

```
    plt.legend(bbox_to_anchor=(1.0, 1.0), loc='upper left')
    plt.show()

# TO DO call plot_scatter with different feature pairs to create some␣
 ↪visualizations

plot_scatter('culmen_length_mm', 'flipper_length_mm')
plot_scatter('culmen_depth_mm', 'flipper_length_mm')
plot_scatter('body_mass_g', 'flipper_length_mm')
```







3b Suppose you want to be able to identify the Gentoo species with a simple rule with very high accuracy. Use a decision tree classifier to figure out such a rule that has only two checks (e.g. "mass greater than 4000 g, and culmen length less than 40 mm is Gentoo; otherwise, not"). You can use the library DecisionTreeClassifier with either 'gini' or 'entropy' criterion. Use sklearn.tree.plot_tree with feature_names and class_names arguments to visualize the decision tree. Include the tree that you used to find the rule in your report and the rule.

[68]:
```
# TO DO (Train a short tree to identify a good rule, plot the tree, report the␣
↪rule and its precision/recall in your report)
X, y, feature_names, class_names = get_penguin_xy(df_penguins)
y = 1*(y =='Gentoo')
from sklearn.tree import DecisionTreeClassifier, plot_tree,export_text
from sklearn.metrics import precision_score, recall_score, confusion_matrix
tree = DecisionTreeClassifier(max_depth=2,max_features=1)
tree.fit(X,y)
```

[68]:
```
DecisionTreeClassifier(max_depth=2, max_features=1)
```

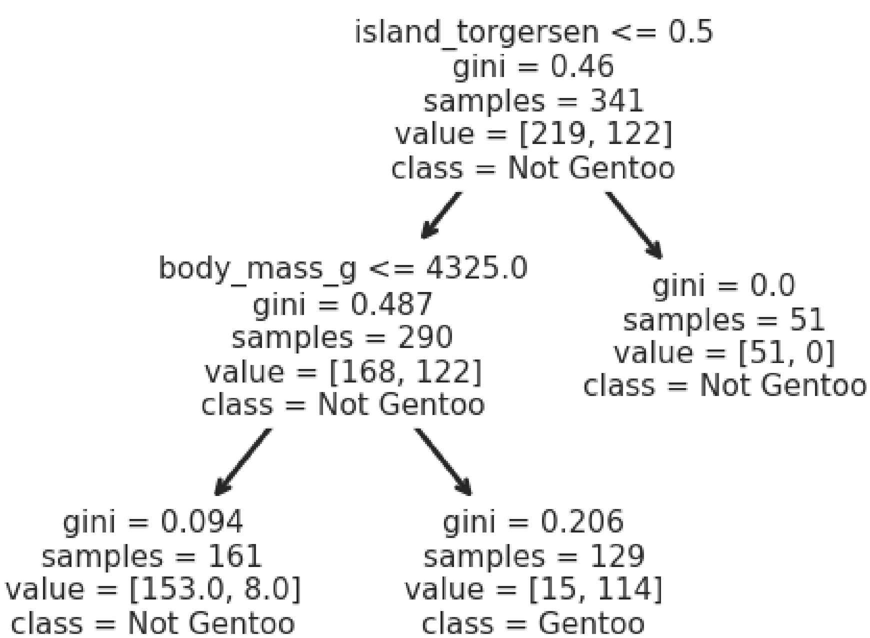
[72]:
```
y_pred = tree.predict(X)
print(f"Precision {precision_score(y,y_pred)}")
print(f"Recall {recall_score(y,y_pred)}")
print(f"Acc {(y_pred==y).mean()}")
print(confusion_matrix(y,y_pred))
print(114 / (114+15))
```

```
print(114 / (114+8))
```

```
Precision 0.8837209302325582
Recall 0.9344262295081968
Acc 0.9325513196480938
[[204  15]
 [  8 114]]
0.8837209302325582
0.9344262295081968
```

[70]:
```
print(export_text(tree, feature_names = feature_names, class_names=['Not Gentoo','Gentoo']))
plot_tree(tree,feature_names=feature_names, class_names=["Not Gentoo", "Gentoo"])
None
```

```
|--- island_torgersen <= 0.50
|   |--- body_mass_g <= 4325.00
|   |   |--- class: Not Gentoo
|   |--- body_mass_g >  4325.00
|   |   |--- class: Gentoo
|--- island_torgersen >  0.50
|   |--- class: Not Gentoo
```



3c Use any method at your disposal to achieve maximum 5-fold cross-validation accuracy on this problem. To keep it simple, we will use sklearn.model_selection to perform the cross-validation for us. Report your model design and 5-fold accuracy. It is possible to get more than 99% accuracy.

[76]:
```
# design a classification model, import libraries as needed
from sklearn.model_selection import cross_val_score

X, y, feature_names, class_names = get_penguin_xy(df_penguins)

# TO DO -- choose some model and fit the data
from sklearn.datasets import load_iris
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import numpy as np

model = GradientBoostingClassifier()



scores = cross_val_score(model, np.array(X), np.array(y), cv=5)
print('CV Accuracy: {}'.format(scores.mean()))
```

```
CV Accuracy: 0.9882352941176471
```

0.3 Part 4: Stretch Goals

Include any new code needed for Part 4 here

1 4a

[35]:
```
# initialization code
import numpy as np
from keras.datasets import mnist
%matplotlib inline
from matplotlib import pyplot as plt
from scipy import stats
import torch
import torch.nn as nn

def load_mnist():
  '''
  Loads, reshapes, and normalizes the data
  '''
  (x_train, y_train), (x_test, y_test) = mnist.load_data() # loads MNIST data
```

```
    x_train = np.reshape(x_train, (len(x_train), 28*28))  # reformat to 768-d␣
↪vectors
    x_test = np.reshape(x_test, (len(x_test), 28*28))
    maxval = x_train.max()
    x_train = x_train/maxval  # normalize values to range from 0 to 1
    x_test = x_test/maxval

    m = x_train.mean()
    s = x_train.std()
    x_train = (x_train-m)/s
    x_test = (x_test-m)/s
    return (x_train, y_train), (x_test, y_test)

def display_mnist(x, subplot_rows=1, subplot_cols=1):
    '''
    Displays one or more examples in a row or a grid
    '''
    if subplot_rows>1 or subplot_cols>1:
      fig, ax = plt.subplots(subplot_rows, subplot_cols, figsize=(15,15))
      for i in np.arange(len(x)):
        ax[i].imshow(np.reshape(x[i], (28,28)), cmap='gray')
        ax[i].axis('off')
    else:
        plt.imshow(np.reshape(x, (28,28)), cmap='gray')
        plt.axis('off')
    plt.show()
```

[36]:
```
# Sets device to "cuda" if a GPU is available  (in Colabs, enable GPU by␣
↪Edit->Notebook Settings-->Hardware Accelerator=GPU)
device = "cuda" if torch.cuda.is_available() else 'cpu'
print(device) # make sure you're using GPU instance
```

```
cuda
```

[37]:
```
(x_train, y_train), (x_test, y_test) = load_mnist()

# create train/val split
ntrain = 50000
x_val = x_train[ntrain:].copy()
y_val = y_train[ntrain:].copy()
x_train = x_train[:ntrain]
y_train = y_train[:ntrain]
x_val.shape
```

[37]:
```
(10000, 784)
```

[38]:
```
def display_error_curves(training_losses, validation_losses):
    """
    Plots the training and validation loss curves
    training_losses and validation_losses should be lists or arrays of the same␣
    ↪length
    """
    num_epochs = len(training_losses)

    plt.plot(range(num_epochs), training_losses, label="Training Loss")
    plt.plot(range(num_epochs), validation_losses, label="Validation Loss")

    # Add in a title and axes labels
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')

    # Display the plot
    plt.legend(loc='best')
    plt.show()
```

[39]:
```
# Define the model
class MLP(nn.Module):
    def __init__(self, input_size, output_size):
        super(MLP, self).__init__()
        h1 = 512
        h2 = 256
        self.l1 = nn.Linear(input_size,h1)
        self.l2 = nn.Linear(h1,h2)
        self.l3 = nn.Linear(h2,output_size)
        self.act = nn.ReLU()

    def forward(self, x):
        x = self.act(self.l1(x))
        x = self.act(self.l2(x))
        x = self.l3(x)
        return x
```

[51]:
```
# This is a possible function definition for training MLP, but feel free to␣
↪change it
# You may also want to create helper functions, e.g. for computing loss or␣
↪prediction
def train_MLP_mnist(train_loader, val_loader, lr=1e-1, num_epochs=100):
    '''
    Train a MLP
    Input: train_loader and val_loader are dataloaders for the training and
    val data, respectively. lr is the learning rate, and the network will
    be trained for num_epochs epochs.
```

```
  Output: return a trained MLP
  '''
  # TODO: fill in all code

  input_size = 784
  output_size = 10

  # Instantiate the model
  global mlp
  mlp = MLP(input_size,output_size)
  mlp.to(device)

  # Train the model, compute and store train/val loss at each epoch
  train_losses,val_losses = [],[]
  optim = torch.optim.SGD(mlp.parameters(),lr=lr)
  for e in range(num_epochs):
    print(f"Epoch: {e}")
    mlp.train()
    tloss, terr = evaluate_MLP(mlp,train_loader,optim)
    mlp.eval()
    vloss, verr = evaluate_MLP(mlp,val_loader,None)
    train_losses.append(tloss)
    val_losses.append(vloss)
    print(f"Terr {100*terr:.2f}   verr {100*verr:.2f}")
    print(f"Tloss {tloss} vloss {vloss}")


  # Display Loss Curves
  display_error_curves(train_losses,val_losses)
  print(np.min(val_losses), np.argmin(val_losses))
  return mlp


def evaluate_MLP(mlp, loader, optim):
  ''' Computes loss and error rate given your mlp model and data loader'''
  N = 0
  acc = 0
  loss = 0
  loss_function = torch.nn.CrossEntropyLoss()
  with torch.set_grad_enabled(optim is not None):
    for i, data in enumerate(loader, 0):
      # print(i)
      # Get inputs
      inputs, targets = data
      N += len(targets)

      # Perform forward pass
```

```
      outputs = mlp(inputs.to(device))

      # Compute sum of correct labels
      y_pred = np.argmax(outputs.detach().cpu().numpy(), axis=1)
      y_gt = np.argmax(targets.numpy(), axis=1)
      acc += np.sum(y_pred==y_gt)

      # Compute loss
      L = loss_function(outputs, targets.to(device))
      if optim:
        optim.zero_grad()
        L.backward()
        optim.step()
      loss += L.item()*len(targets)


  loss /= N
  acc /= N

  return loss, 1-acc
```

[61]:
```
# Code for running experiments

print(device) # make sure you're using GPU instance
torch.manual_seed(0) # to avoid randomness, but if you wanted to create an␣
 ↪ensemble, you should not use a manual seed

# TODO (set up dataloaders, and call training function)
trainset = torch.utils.data.TensorDataset(torch.Tensor(x_train), torch.
 ↪Tensor(np.eye(10)[y_train]))
train_loader = torch.utils.data.DataLoader(trainset, batch_size=64,␣
 ↪shuffle=True, num_workers=0)

valset = torch.utils.data.TensorDataset(torch.Tensor(x_val), torch.Tensor(np.
 ↪eye(10)[y_val]))
val_loader = torch.utils.data.DataLoader(valset, batch_size=1000, shuffle=True,␣
 ↪num_workers=0)
```

cuda

[62]:
```
testset = torch.utils.data.TensorDataset(torch.Tensor(x_test), torch.Tensor(np.
 ↪eye(10)[y_test]))
test_loader = torch.utils.data.DataLoader(testset, batch_size=1000,␣
 ↪shuffle=True, num_workers=0)
```

[63]:
```
train_MLP_mnist(train_loader,val_loader,0.05)
```

```
Epoch: 0
Terr 11.10   verr 5.24
Tloss 0.39561834798812867 vloss 0.18281921446323396
Epoch: 1
Terr 4.77   verr 4.52
Tloss 0.1595677501678467 vloss 0.14803548008203507
Epoch: 2
Terr 3.15   verr 3.02
Tloss 0.10630839919507504 vloss 0.10362636521458626
Epoch: 3
Terr 2.30   verr 2.59
Tloss 0.0775736337813735 vloss 0.08799983635544777
Epoch: 4
Terr 1.64   verr 2.45
Tloss 0.057506488343775274 vloss 0.08281317055225372
Epoch: 5
Terr 1.29   verr 2.18
Tloss 0.04498154129862785 vloss 0.0753883071243763
Epoch: 6
Terr 0.96   verr 2.40
Tloss 0.034889506816864015 vloss 0.07816188521683216
Epoch: 7
Terr 0.67   verr 5.23
Tloss 0.02670353217050433 vloss 0.18872876614332199
Epoch: 8
Terr 0.52   verr 3.63
Tloss 0.02145877429395914 vloss 0.1292068473994732
Epoch: 9
Terr 0.35   verr 2.42
Tloss 0.01578059840232134 vloss 0.08922246024012566
Epoch: 10
Terr 0.23   verr 2.01
Tloss 0.011982590095708146 vloss 0.07129322849214077
Epoch: 11
Terr 0.12   verr 1.91
Tloss 0.0089017694882676 vloss 0.07053324021399021
Epoch: 12
Terr 0.10   verr 1.81
Tloss 0.00725795012280345 vloss 0.06977238543331624
Epoch: 13
Terr 0.05   verr 1.81
Tloss 0.005373965505324304 vloss 0.07086622230708599
Epoch: 14
Terr 0.03   verr 1.77
Tloss 0.004175682089366019 vloss 0.06942166555672884
Epoch: 15
Terr 0.02   verr 1.68
Tloss 0.0034479063447006046 vloss 0.06911577358841896
```

```
Epoch: 16
Terr 0.01   verr 1.74
Tloss 0.002948415506063029 vloss 0.07080496475100517
Epoch: 17
Terr 0.01   verr 1.75
Tloss 0.0024644659453071653 vloss 0.07182458527386189
Epoch: 18
Terr 0.01   verr 1.70
Tloss 0.0022004828157415612 vloss 0.07197846844792366
Epoch: 19
Terr 0.00   verr 1.67
Tloss 0.0018988471547781956 vloss 0.07276872247457504
Epoch: 20
Terr 0.00   verr 1.73
Tloss 0.0017089839291479438 vloss 0.07219697572290898
Epoch: 21
Terr 0.00   verr 1.62
Tloss 0.0015370161229558289 vloss 0.07341913431882859
Epoch: 22
Terr 0.00   verr 1.66
Tloss 0.0013948519614432008 vloss 0.07381088398396969
Epoch: 23
Terr 0.00   verr 1.68
Tloss 0.0012932354394404684 vloss 0.0748615387827158
Epoch: 24
Terr 0.00   verr 1.68
Tloss 0.001173391508362256 vloss 0.07537885643541813
Epoch: 25
Terr 0.00   verr 1.66
Tloss 0.0010890333968913183 vloss 0.0752104852348566
Epoch: 26
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[63], line 1
----> 1 train_MLP_mnist(train_loader,val_loader,0.05)

Cell In[51], line 27, in train_MLP_mnist(train_loader, val_loader, lr,␣
 ↪num_epochs)
     25 print(f"Epoch: {e}")
     26 mlp.train()
---> 27 tloss, terr = evaluate_MLP(mlp,train_loader,optim)
     28 mlp.eval()
     29 vloss, verr = evaluate_MLP(mlp,val_loader,None)

Cell In[51], line 56, in evaluate_MLP(mlp, loader, optim)
     53 N += len(targets)
```

```
     55 # Perform forward pass
---> 56 outputs = mlp(inputs.to(device))
     58 # Compute sum of correct labels
     59 y_pred = np.argmax(outputs.detach().cpu().numpy(), axis=1)

File ~/.local/lib/python3.11/site-packages/torch/nn/modules/module.py:1518, in Module._wrapped_call_impl(self, *args, **kwargs)
   1516     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1517 else:
-> 1518     return self._call_impl(*args, **kwargs)

File ~/.local/lib/python3.11/site-packages/torch/nn/modules/module.py:1527, in Module._call_impl(self, *args, **kwargs)
   1522 # If we don't have any hooks, we want to skip the rest of the logic in
   1523 # this function, and just call forward.
   1524 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1525         or _global_backward_pre_hooks or _global_backward_hooks
   1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
   1529 try:
   1530     result = None

Cell In[39], line 14, in MLP.forward(self, x)
     12 def forward(self, x):
     13     x = self.act(self.l1(x))
---> 14     x = self.act(self.l2(x))
     15     x = self.l3(x)
     16     return x

File ~/.local/lib/python3.11/site-packages/torch/nn/modules/module.py:1518, in Module._wrapped_call_impl(self, *args, **kwargs)
   1516     return self._compiled_call_impl(*args, **kwargs)  # type: ignore[misc]
   1517 else:
-> 1518     return self._call_impl(*args, **kwargs)

File ~/.local/lib/python3.11/site-packages/torch/nn/modules/module.py:1527, in Module._call_impl(self, *args, **kwargs)
   1522 # If we don't have any hooks, we want to skip the rest of the logic in
   1523 # this function, and just call forward.
   1524 if not (self._backward_hooks or self._backward_pre_hooks or self._forward_hooks or self._forward_pre_hooks
   1525         or _global_backward_pre_hooks or _global_backward_hooks
   1526         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1527     return forward_call(*args, **kwargs)
   1529 try:
```

```
   1530     result = None

File ~/.local/lib/python3.11/site-packages/torch/nn/modules/linear.py:114, in␣
↪Linear.forward(self, input)
    113 def forward(self, input: Tensor) -> Tensor:
--> 114     return F.linear(input, self.weight, self.bias)

KeyboardInterrupt:
```

[67]:
```
test_loss, test_err = evaluate_MLP(mlp,val_loader,None)
print(test_loss, 100*test_err)
```

```
0.07519146390259265 1.6100000000000003
```

[ ]:
```
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/My Drive/CS441/24SP/hw2" # @param {type:
↪"string"}
NOTEBOOK_NAME = "CS441_SP24_HW2_Solution.ipynb" # @param {type:"string"}
#------------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic >␣
↪/dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```