CS441_HW4_Solution

November 3, 2025

0.1 CS441: Applied ML - HW 4

0.1.1 Part 1: Model Complexity and Tree-based Regressors

[ ]:
```
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/MyDrive/CS441/hw1/"
  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()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

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:

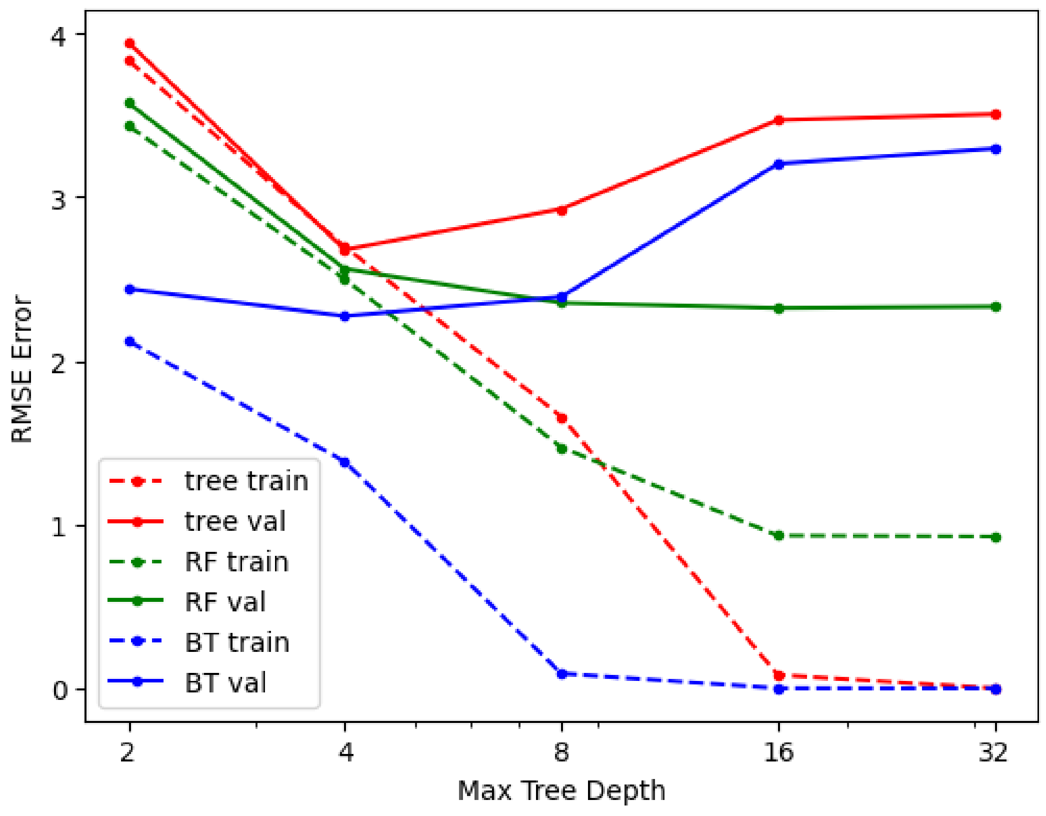
- max_depth={2,4,8,16,32}
- random_state=0
- For random forest: max_features=1/3

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.

[ ]:
```
# 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()
```

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

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

# usage examples
# model = DecisionTreeRegressor(random_state=0, max_depth=max_depth)
```

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

dtr_trainerror = []
dtr_valerror = []
rfr_trainerror = []
rfr_valerror = []
gbr_trainerror = []
gbr_valerror = []

for max_depth in max_depths:
  # DecisionTreeRegressor
  model_dtr = DecisionTreeRegressor(random_state = 0, max_depth= max_depth)
  model_dtr.fit(x_train,y_train)
  dtr_train_ypredict = model_dtr.predict(x_train)
  dtr_val_ypredict = model_dtr.predict(x_val)
  dtr_trainerror.append(np.sqrt(mean_squared_error(y_train,dtr_train_ypredict)))
  dtr_valerror.append(np.sqrt(mean_squared_error(y_val,dtr_val_ypredict)))

  # RandomForestRegressor
  model_rfr = RandomForestRegressor(random_state = 0, max_depth= max_depth)
  model_rfr.fit(x_train,y_train)
  rfr_train_ypredict = model_rfr.predict(x_train)
  rfr_val_ypredict = model_rfr.predict(x_val)
  rfr_trainerror.append(np.sqrt(mean_squared_error(y_train,rfr_train_ypredict)))
  rfr_valerror.append(np.sqrt(mean_squared_error(y_val,rfr_val_ypredict)))

  # GradientBoostingRegressor
  model_gbr = GradientBoostingRegressor(random_state = 0, max_depth= max_depth)
  model_gbr.fit(x_train,y_train)
  gbr_train_ypredict = model_gbr.predict(x_train)
  gbr_val_ypredict = model_gbr.predict(x_val)
  gbr_trainerror.append(np.sqrt(mean_squared_error(y_train,gbr_train_ypredict)))
  gbr_valerror.append(np.sqrt(mean_squared_error(y_val,gbr_val_ypredict)))

plot_depth_error(max_depths, dtr_trainerror, dtr_valerror, rfr_trainerror,␣
↪rfr_valerror, gbr_trainerror, gbr_valerror)
```



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.

[ ]:
```
# 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
  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()
```

[ ]:
```
# 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.

[ ]:
```
(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]
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434          2s
0us/step
```

[ ]:
```
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()
```

[ ]:
```
# Define the model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        # Needs code here
        self.model = nn.Sequential(
            nn.Linear(input_size,hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, output_size)
        )

    def forward(self, x):
      # Needs code here
        return self.model(x)
```

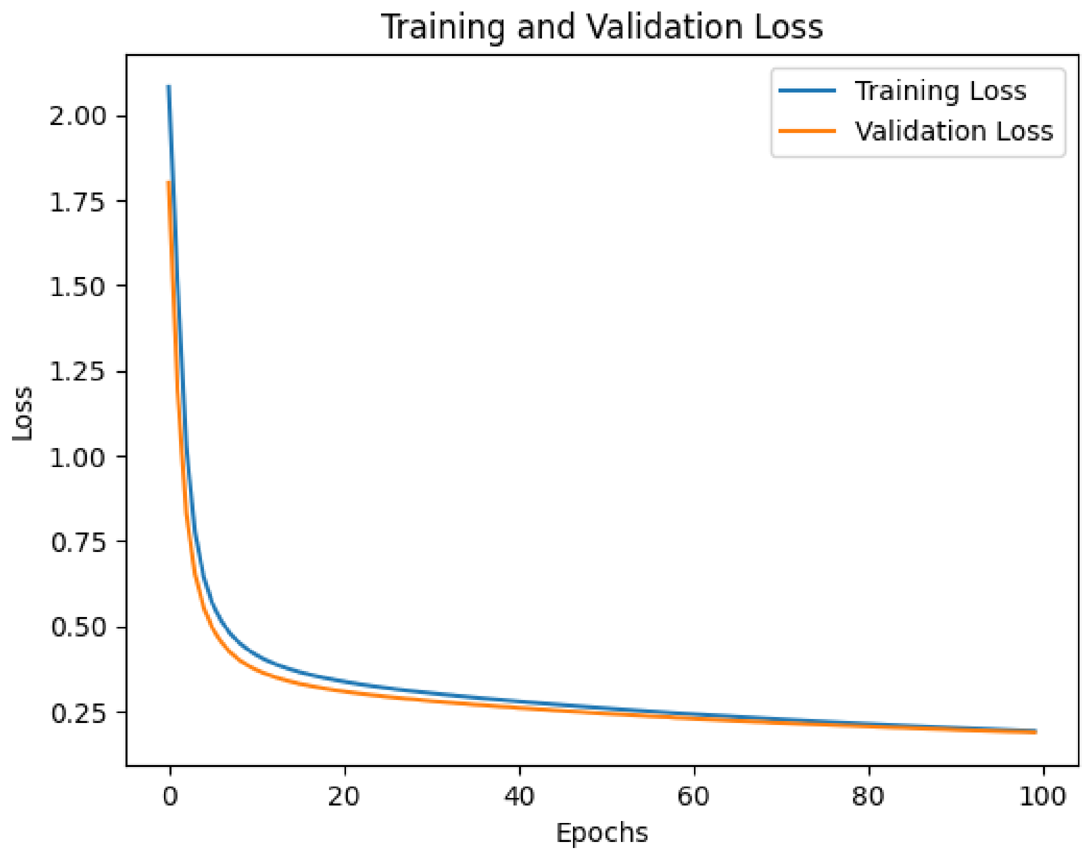
[ ]:
```
print(x_train.shape)
```

```
(50000, 784)
```

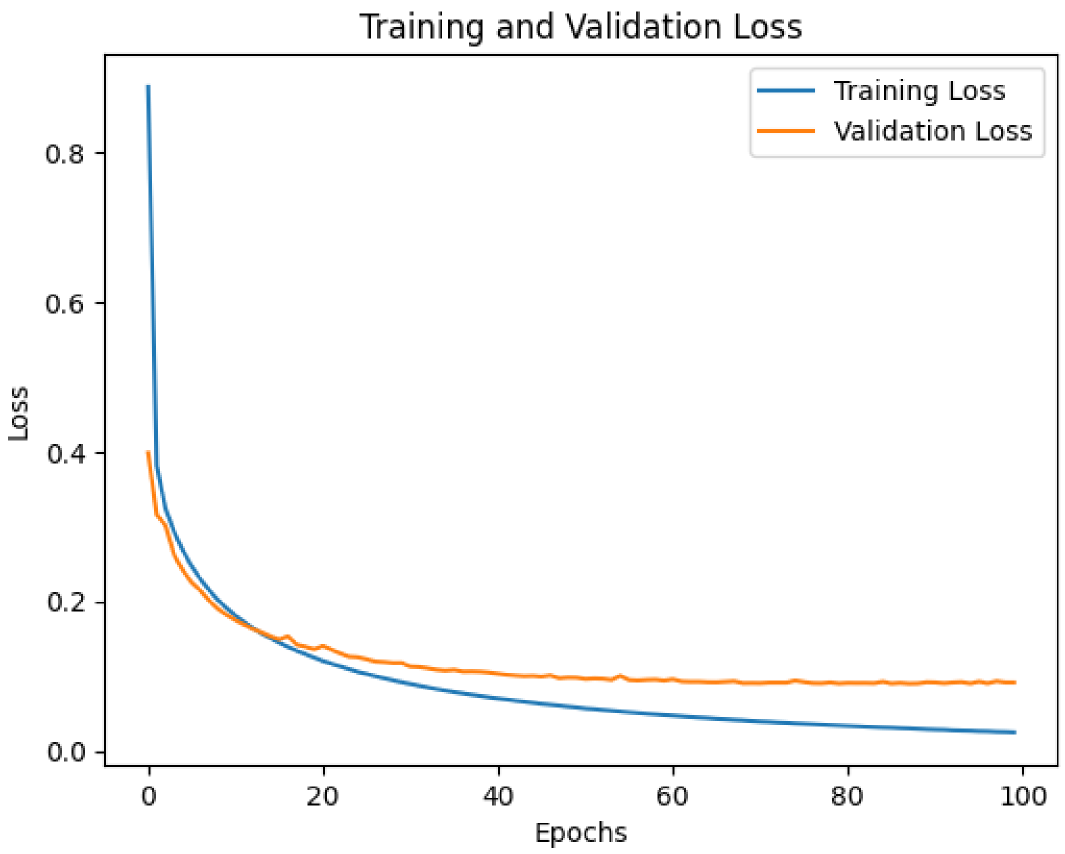
```
[ ]: # This is a possible function definition for training MLP, but feel free to␣
      ↪change it
     # You may also want to create helper functions, e.g. for computing loss or␣
      ↪prediction
     def train_MLP_mnist(train_loader, val_loader, lr=1e-1, num_epochs=100):
       '''
       Train 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 = x_train.shape[1]
       hidden_size = 64
       output_size = 10

       # Instantiate the model
       mlp = MLP(input_size,hidden_size,output_size).to(device)
       criterion = torch.nn.CrossEntropyLoss()
       optimizer = torch.optim.SGD(mlp.parameters(),lr=lr) # 0.01, 0.1, 1
       train_loss_list = []
       val_loss_list = []
       val_loss_epochmin = 0

       # Train the model, compute and store train/val loss at each epoch

       epochs = 100
       for epoch in range(epochs):
         mlp.train()
         running_loss = 0
         n = 0
         for inputs,targets in train_loader:
           inputs = inputs.to(device)
           targets = targets.to(device)

           outputs = mlp(inputs)
           loss = criterion(outputs,targets)

           optimizer.zero_grad()
           loss.backward()
           optimizer.step()

           # loss.item   batch
           running_loss += loss.item()*len(targets)  # last batch   256
           n += len(targets) # sample numbers of all epoch

         train_loss_list.append(running_loss/n) # loss of an epoch

         mlp.eval()
         val_loss, val_error = evaluate_MLP(mlp,val_loader)
         val_loss_list.append(val_loss)

       val_loss_epochmin = np.argmin(val_loss_list)
       print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss_list[-1]:.
     ↪4f}, Val Loss: {val_loss_list[-1]:.4f}")
       print("Best_Epoch:",val_loss_epochmin)

       # Display Loss Curves
       display_error_curves(train_loss_list, val_loss_list)

       return mlp


     def evaluate_MLP(mlp, loader):
       ''' 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(False):
         for i, data in enumerate(loader, 0):

           # 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.cpu().numpy(), axis=1)
           y_gt = targets.numpy()
           acc += np.sum(y_pred==y_gt)

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

       loss /= N
       acc /= N
       return loss, 1-acc
```

```
[ ]: from torch.utils.data import DataLoader, TensorDataset
     # 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)
     x_train_tensor = torch.tensor(x_train, dtype=torch.float32)
     y_train_tensor = torch.tensor(y_train, dtype=torch.long)

     x_val_tensor = torch.tensor(x_val, dtype=torch.float32)
     y_val_tensor = torch.tensor(y_val, dtype=torch.long)

     batch_size = 256
     train_loader =␣
      ↪DataLoader(TensorDataset(x_train_tensor,y_train_tensor),batch_size =␣
      ↪256,shuffle=True)
     val_loader = DataLoader(TensorDataset(x_val_tensor,y_val_tensor),batch_size =␣
      ↪256,shuffle=False)
     lrs = [0.01,0.1,1]

     for lr in lrs:
       model = train_MLP_mnist(train_loader, val_loader, lr=lr)
       evaluate_MLP(model,val_loader)
```

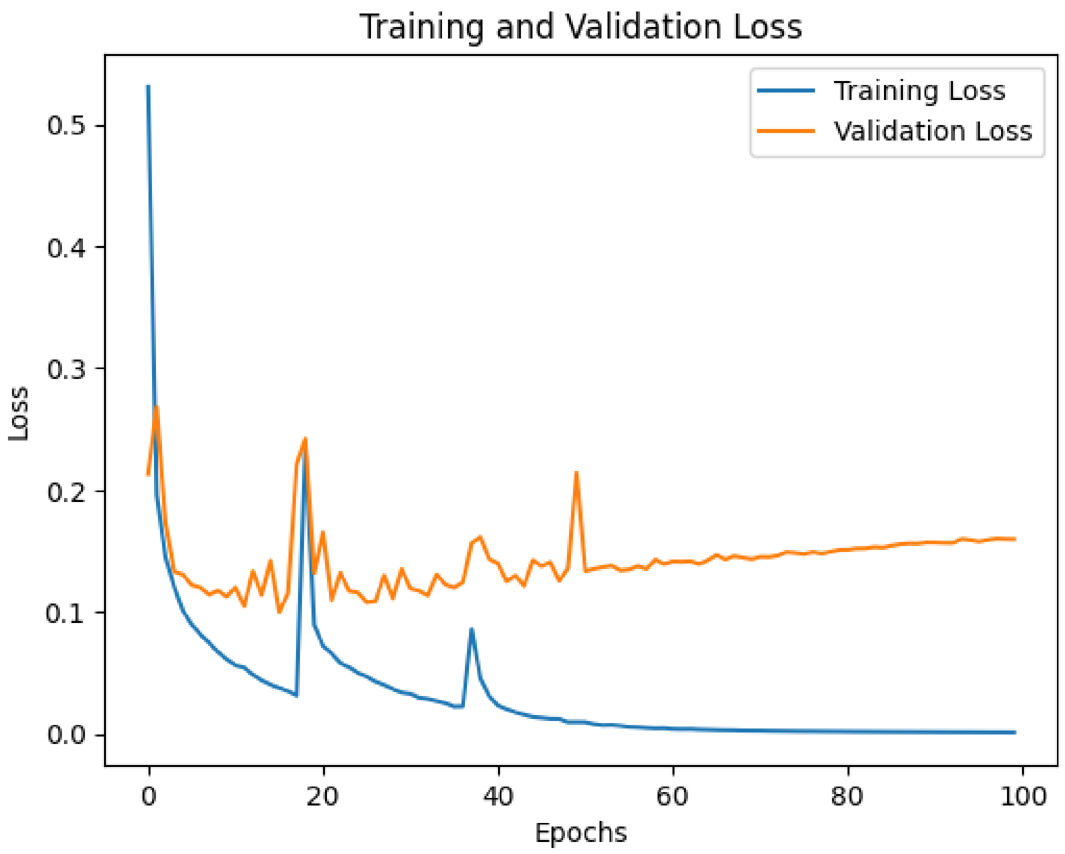
```
cuda
Epoch [100/100], Train Loss: 0.1909, Val Loss: 0.1873
Best_Epoch: 99
```



```
Epoch [100/100], Train Loss: 0.0250, Val Loss: 0.0916
Best_Epoch: 87
```



```
Epoch [100/100], Train Loss: 0.0011, Val Loss: 0.1597
Best_Epoch: 15
```



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.

[ ]:
```
# TO DO (retrain if needed, and evaluate model on train, val, and test sets)
best_lr = 0.1
best_epoch = 87

best_model = MLP(x_train.shape[1], 64, 10).to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(best_model.parameters(), lr=best_lr)

for epoch in range(best_epoch):
  best_model.train()
  for inputs,targets in train_loader:
    inputs = inputs.to(device)
    targets = targets.to(device)

    optimizer.zero_grad()
    outputs = best_model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()

    optimizer.step()

x_test_tensor = torch.tensor(x_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)
test_loader =␣
  ↪DataLoader(TensorDataset(x_test_tensor,y_test_tensor),batch_size=256,shuffle=False)

train_loss, train_error = evaluate_MLP(best_model, train_loader)
val_loss, val_error = evaluate_MLP(best_model, val_loader)
_, test_error = evaluate_MLP(best_model, test_loader)

print(f"Training Loss: {train_loss:.1E}, Training Error: {train_error:.4f}")
print(f"Validation Loss: {val_loss:.1E}, Validation Error: {val_error:.4f}")
print(f"Test Error: {test_error:.4f}")
```

```
Training Loss: 2.9E-02, Training Error: 0.0061
Validation Loss: 8.8E-02, Validation Error: 0.0239
Test Error: 0.0257
```

0.2 Part 3: Predicting Penguin Species

Include all your code for part 3 in this section.

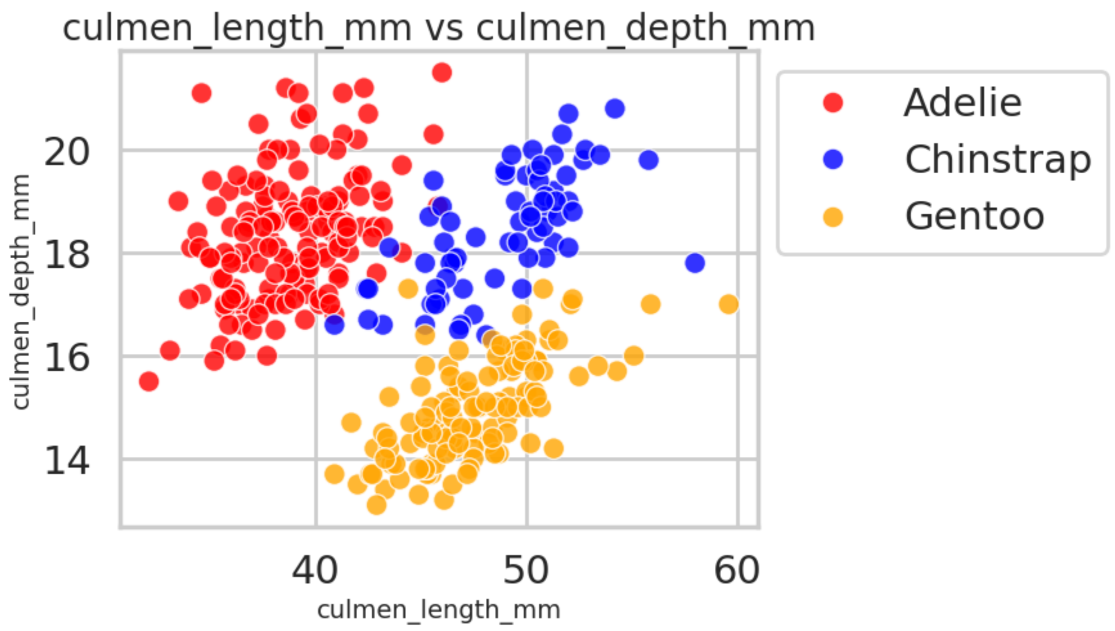
```
[ ]: import numpy as np
from google.colab import drive
%matplotlib inline
from matplotlib import pyplot as plt
import pandas as pd
import seaborn as sns
#styling preferences for sns
sns.set_style('whitegrid')
sns.set_context('poster')
drive.mount('/content/gdrive/')
datadir = "/content/gdrive/MyDrive/CS441/hw4/"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

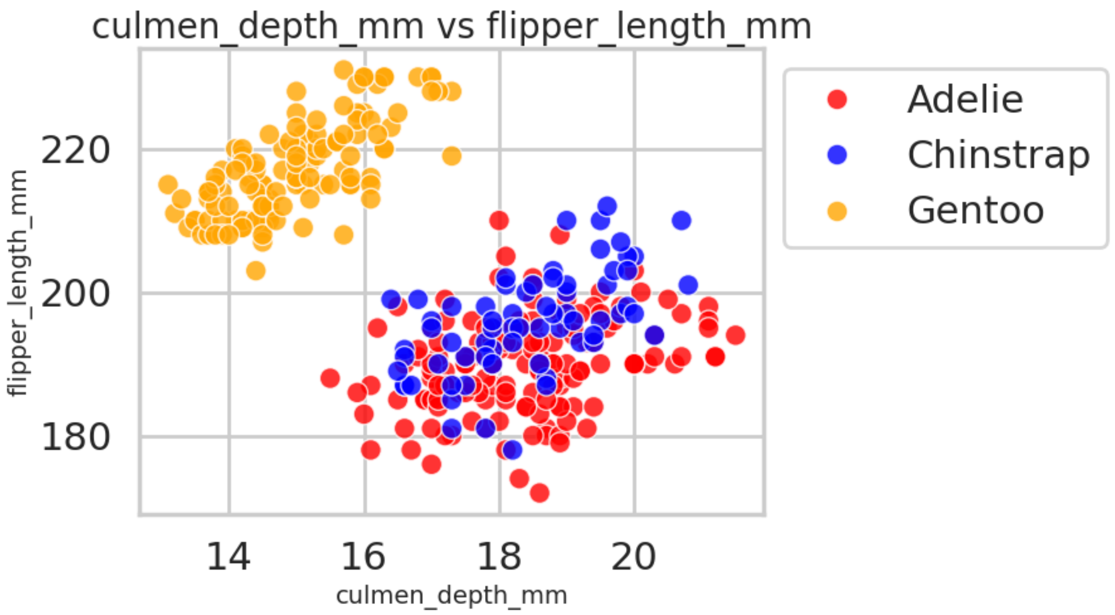
```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call
drive.mount("/content/gdrive/", force_remount=True).
```

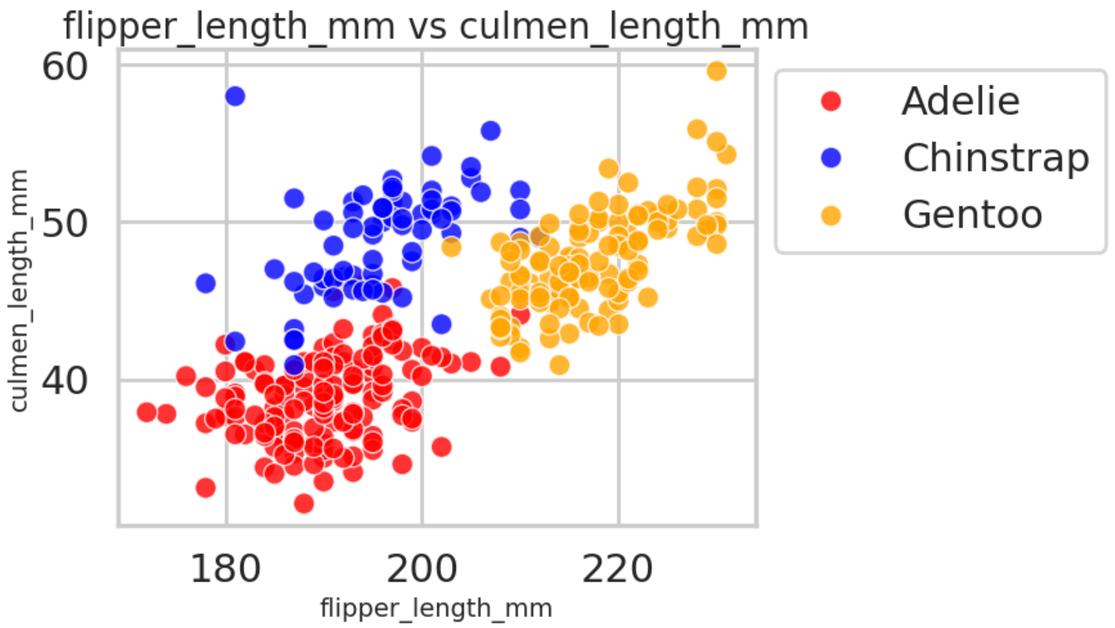
3a Spend some time to visualize different pairs of features and their relationships to the species. We've done one for you. Include in your report at least two other visualizations.

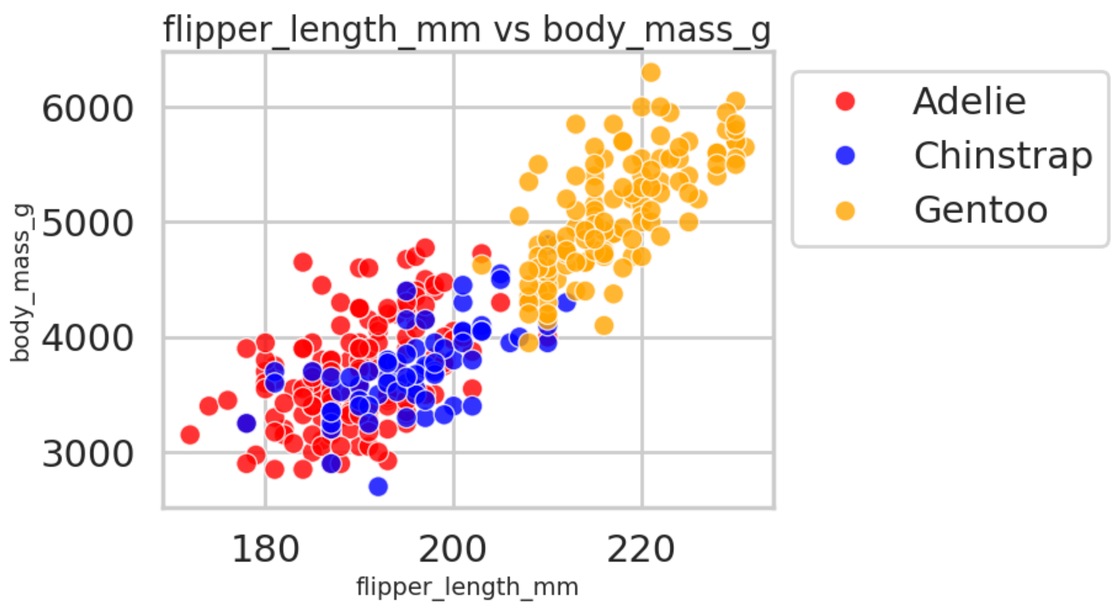
```
[ ]: def plot_scatter(feature1, feature2):
       '''
       Provide names of two features to create a scatterplot of them
       E.g. plot_scatter('culmen_length_mm', 'culmen_depth_mm')
       Possible features: 'culmen_length_mm', 'culmen_depth_mm',␣
     ↪'flipper_length_mm', 'body_mass_g'
  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

       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', 'culmen_depth_mm')

# Visualization 1
plot_scatter('culmen_depth_mm', 'flipper_length_mm')

# Visualization 2
plot_scatter('flipper_length_mm', 'culmen_length_mm')

# Visualization 3
plot_scatter('flipper_length_mm', 'body_mass_g')
```




culmen_depth_mm vs flipper_length_mm
flipper_length_mm
220
200
180
14
16
18
20
culmen_depth_mm
Adelie
Chinstrap
Gentoo

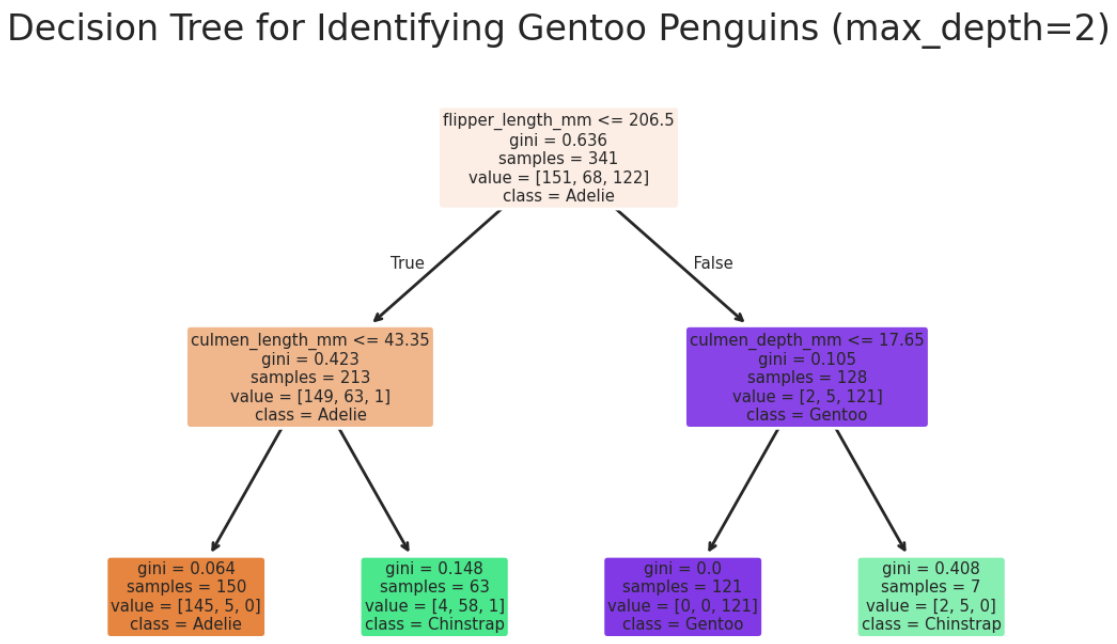

flipper_length_mm vs culmen_length_mm
culmen_length_mm
60
50
40
180
200
220
flipper_length_mm
Adelie
Chinstrap
Gentoo



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.

```
[ ]: # TO DO (Train a short tree to identify a good rule, plot the tree, report the␣
      ↪rule and its precision/recall in your report)
     from sklearn.tree import DecisionTreeClassifier, plot_tree
     from sklearn.metrics import classification_report, precision_score, recall_score
     import matplotlib.pyplot as plt

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

     model = DecisionTreeClassifier(max_depth = 2, criterion='gini', random_state=0)
     model.fit(X,y)

     plt.figure(figsize=(12, 8))
     plot_tree(model, feature_names=feature_names, class_names=class_names,␣
      ↪filled=True, rounded=True)
     plt.title("Decision Tree for Identifying Gentoo Penguins (max_depth=2)")
     plt.show()

     y_pred = model.predict(X)
     print(classification_report(y, y_pred))
```

```
['island_biscoe', 'island_dream', 'island_torgersen', 'culmen_length_mm',
'culmen_depth_mm', 'flipper_length_mm', 'body_mass_g', 'sex_female', 'sex_male',
'sex_unknown'] ['Adelie' 'Chinstrap' 'Gentoo']
```



```
              precision    recall  f1-score   support

      Adelie       0.97      0.96      0.96       151
   Chinstrap       0.90      0.93      0.91        68
      Gentoo       1.00      0.99      1.00       122

    accuracy                           0.96       341
   macro avg       0.96      0.96      0.96       341
weighted avg       0.97      0.96      0.97       341
```

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.

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

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

```
# TO DO -- choose some model and fit the data
model = RandomForestClassifier( n_estimators=200,max_depth=None,random_state=0
)

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

```
CV Accuracy: 0.9911764705882353
```

0.3 Part 4: Stretch Goals

Include any new code needed for Part 4 here

```
[ ]: # TO DO (optional)
```

```
[ ]: # 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/MyDrive/CS441/hw4" # @param {type:"string"}
     NOTEBOOK_NAME = "CS441_HW4_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}")
```

```
Mounted at /content/drive/
```