CS273A Homework 5

Due: Wednesday Nov 20 2024 (11:59pm)

Instructions

This homework (and subsequent ones) will involve data analysis and reporting on methods and results using Python code. You will submit a **single PDF file** that contains everything to Gradescope. This includes any text you wish to include to describe your results, the complete code snippets of how you attempted each problem, any figures that were generated, and scans of any work on paper that you wish to include. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it.

Your homeworks will be given to you as Jupyter notebooks containing the problem descriptions and some template code that will help you get started. You are encouraged to use these starter Jupyter notebooks to complete your assignment and to write your report. This will help you not only ensure that all of the code for the solutions is included, but also will provide an easy way to export your results to a PDF file (for example, doing *print preview* and *printing to pdf*). I recommend liberal use of Markdown cells to create headers for each problem and sub-problem, explaining your implementation/answers, and including any mathematical equations. For parts of the homework you do on paper, scan it in such that it is legible (there are a number of free Android/iOS scanning apps, if you do not have access to a scanner), and include it as an image in the Jupyter notebook.

Double check that all of your answers are legible on Gradescope, e.g. make sure any text you have written does not get cut off.

If you have any questions/concerns about using Jupyter notebooks, ask us on EdD. If you decide not to use Jupyter notebooks, but go with Microsoft Word or LaTeX to create your PDF file, make sure that all of the answers can be generated from the code snippets included in the document.

Summary of Assignment: 100 total points

- Problem 1: Decision Trees by Hand (25 points)
  - Problem 1.1: Shannon Entropy (5 points)
  - Problem 1.2: Information Gain (10 points)
  - Problem 1.3: Full Tree (10 points)
- Problem 2: Decision Trees in Python (34 points)
  - Problem 2.1: Feature Statistics (8 points)
  - Problem 2.2: Initial Tree (6 points)
  - Problem 2.3: Exploring Depth Control (10 points)
  - Problem 2.4: Exploring Leaf Size (10 points)
- Problem 3: Random Forests (20 points)
  - Problem 3.1: Training Members (10 points)
  - Problem 3.2: Ensemble Prediction (10 points)
- Problem 4: VC Dimension (16 points)
  - Problem 4.1: Model A (4 points)
  - Problem 4.2: Model B (4 points)
  - Problem 4.3: Model C (4 points)
  - Problem 4.4: Model D (4 points)
- Statement of Collaboration (5 points)

Before we get started, let's import some libraries that you will make use of in this assignment. Make sure that you run the code cell below in order to import these libraries.

Important: In the code block below, we set `seed=1234`. This is to ensure your code has reproducible results and is important for grading. Do not change this. If you are not using the provided Jupyter notebook, make sure to also set the random seed as below.

Important: Do not change any codes we give you below, except for those waiting for you to complete. This is to ensure your code has reproducible results and is important for grading.

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

import requests                                    # reading data
```

```
from io import StringIO

from sklearn.datasets import fetch_openml            # common data set access
from sklearn.preprocessing import StandardScaler     # scaling transform
from sklearn.model_selection import train_test_split # validation tools
from sklearn.metrics import zero_one_loss as J01

import sklearn.tree as tree

# Fix the random seed for reproducibility
# !! Important !! : do not change this
seed = 1234
np.random.seed(seed)
```

Problem 1: Decision Trees for Spam

In order to reduce my email load, I decide to implement a machine learning algorithm to decide whether or not I should read an email, or simply file it away instead. To train my model, I obtain the following data set of binary-valued features about each email, including whether I know the author or not, whether the email is long or short, and whether it has any of several key words, along with my final decision about whether to read it ($y = +1$ for "read", $y = -1$ for "discard").

X1	X2	X3	X4	X5	y
(know author?)	(is long?)	(has 'research'?)	(has 'grade'?)	(has 'lottery'?)	(read?)
0	0	1	1	0	-1
1	1	0	1	0	-1
0	1	1	1	1	-1
1	1	1	1	0	-1
0	1	0	0	0	-1
1	0	1	1	1	1
0	0	1	0	0	1

X1	X2	X3	X4	X5	y
1	0	0	0	0	1
1	0	1	1	0	1
1	1	1	1	1	-1

In the case of any ties where both classes have equal probability, we will prefer to predict class $+1$.

Solve the following problems "by hand" (you can use python for logarithms, etc.)

Problem 1.1 (5 points)

Calculate the Shannon entropy $H(y)$ of the binary class variable y, in bits. **Hint:** Your answer should be a number between 0 and 1.

To calculate the Shannon entropy $H(y)$ of the binary class variable y, we use the formula:

$$H(y) = -\sum_i P(y=i)\log_2 P(y=i)$$

where $P(y=+1)$ and $P(y=-1)$ are the probabilities of the classes $y=+1$ (read) and $y=-1$ (discard), respectively.

Thus,

$$P(y=+1) = \frac{4}{10} = 0.4$$

$$P(y=-1) = \frac{6}{10} = 0.6$$

Then we have:

$$H(y) = -\left(P(y=+1)\log_2 P(y=+1) + P(y=-1)\log_2 P(y=-1)\right)$$

$$H(y) = -\left(0.4\log_2 0.4 + 0.6\log_2 0.6\right) = 0.971$$

We can use the following Python code to calculate the exact number.

In [10]:
```python
import math

# Probabilities
p_y_positive = 4 / 10
p_y_negative = 6 / 10

# Shannon entropy calculation
entropy_y = - (p_y_positive * math.log2(p_y_positive) + p_y_negative * math.log2(p_y_negative))
entropy_y
```

Out[10]:
```
0.9709505944546686
```

Problem 1.2 (5 points)

Calculate the information gain for each feature x_i. Which feature should be split first?

To calculate the information gain $IG(X_i)$ for each feature X_i, we use the formula:

$$IG(X_i) = H(y) - H(y|X_i)$$

where $H(y)$ is the entropy of y (calculated previously as approximately 0.971 bits). $H(y|X_i)$ is the conditional entropy of y given X_i, defined as:

$$H(y|X_i) = P(X_i = 0)H(y|X_i = 0) + P(X_i = 1)H(y|X_i = 1)$$

For Feature X_1, we have

$$P(X_1 = 0) = \frac{4}{10} = 0.4$$

$$P(X_1 = 1) = \frac{6}{10} = 0.6$$

Since

$$H(y|X_1 = 0) = -\sum P(y|X_1 = 0) \cdot \log_2(P(y|X_1 = 0))$$

Thus we have:

$$H(y|X_1 = 0) = -\left(P(y = +1|X_1 = 0) \cdot \log_2(P(y = +1|X_1 = 0)) + P(y = -1|X_1 = 0) \cdot \log_2(P(y = -1|X_1 = 0))\right)$$

$$H(y|X_1 = 0) = -\left(\frac{1}{4} \cdot \log_2(\frac{1}{4}) + \frac{3}{4} \cdot \log_2(\frac{3}{4})\right) = 0.8115$$

We can also calculate $H(y|X_1 = 1)$ in the same way:

$$H(y|X_1 = 1) = -\left(0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)\right) = 1$$

Then we can obtain

$$H(y|X_1) = 0.4 \times 0.8115 + 0.6 \times 1 = 0.9246$$

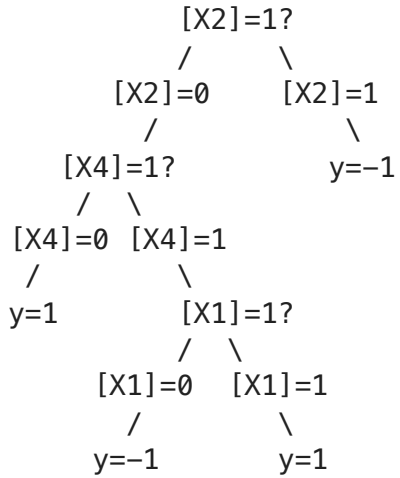
$$IG(X_1) = 0.971 - 0.9246 = 0.0464$$

Similarly, The information gain of other features are $IG(X_2) = 0.61$, $IG(X_3) = 0.0058$, $IG(X_4) = 0.0913$ and $IG(X_5) = 0.0058$. The feature with the highest information gain is X_2 (is the email long?), so we should split on X_2 first.

Problem 1.3 (5 points)

Draw (or otherwise illustrate) **the complete decision tree** that will be learned from these data.

Based on the information gain calculations, we first split on feature X_2, and then continue splitting on features X_5 and X_4. The resulting decision tree is as follows:



- **Root Node**: The tree first splits on X_2 because it has the highest information gain (0.61).
  - If $X_2 = 1$, , the prediction is $y = -1$.
  - If $X_2 = 0$, the tree splits on X_4.
    - If $X_4 = 0$, the prediction is $y = 1$.
    - If $X_4 = 1$, the tree splits on X_1.
      - If $X_1 = 0$, the prediction is $y = -1$.
      - If $X_1 = 1$, the prediction is $y = 1$.

In [56]:

```
import numpy as np
import pandas as pd

# Function to calculate entropy
def entropy(y):
    class_probs = y.value_counts(normalize=True)
    return -np.sum(class_probs * np.log2(class_probs))

# Function to calculate information gain for a feature
def information_gain(X, y, feature):
    # Calculate the entropy of the target variable (H(y))
    H_y = entropy(y)

    # Split data based on feature values and calculate the weighted entropy of the subsets
    feature_values = X[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset_y = y[X[feature] == value]
        weighted_entropy += (len(subset_y) / len(y)) * entropy(subset_y)

    # Information Gain = H(y) - Weighted Entropy of Subsets
    return H_y - weighted_entropy

# Create the dataset
data = {
    'X1': [0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
    'X2': [0, 1, 1, 1, 1, 0, 0, 0, 0, 1],
    'X3': [1, 0, 1, 1, 0, 1, 1, 0, 1, 1],
    'X4': [1, 1, 1, 1, 0, 1, 0, 0, 1, 1],
    'X5': [0, 0, 1, 0, 0, 1, 0, 0, 0, 1],
    'y': [-1, -1, -1, -1, -1, 1, 1, 1, 1, -1]
}

# Convert to DataFrame
df = pd.DataFrame(data)

# Features (X) and labels (y)
X = df.drop('y', axis=1)
y = df['y']
```

```
# Calculate Information Gain for each feature
for feature in X.columns:
    ig = information_gain(X, y, feature)
    print(f"Information Gain for {feature}: {ig:.4f}")
```

```
Information Gain for X1: 0.0464
Information Gain for X2: 0.6100
Information Gain for X3: 0.0058
Information Gain for X4: 0.0913
Information Gain for X5: 0.0058
```

Problem 2: Decision trees in Python

In the next problem, we will use decision trees to predict a data set used for Kaggle competitions in older iterations of the course; see e.g., https://www.kaggle.com/c/uc-irvine-cs273a-2016

Note that I have altered the data from that competition to make the target variables +1 and -1, instead of 0/1, to better match some of the ensemble slides.

First, let's load the data:

In [71]:
```
url = 'https://www.ics.uci.edu/~ihler/classes/cs273/data/precip_train.txt'

with requests.get(url, verify=False) as link:
  data = np.genfromtxt(StringIO(link.text),delimiter=None)

X,Y = data[:,:-1], data[:,-1]
```

```
/opt/anaconda3/lib/python3.12/site-packages/urllib3/connectionpool.py:1099: InsecureRequestWarning: Unverified HTTPS
request is being made to host 'www.ics.uci.edu'. Adding certificate verification is strongly advised. See: https://u
rllib3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/urllib3/connectionpool.py:1099: InsecureRequestWarning: Unverified HTTPS
request is being made to host 'ics.uci.edu'. Adding certificate verification is strongly advised. See: https://urlli
b3.readthedocs.io/en/latest/advanced-usage.html#tls-warnings
  warnings.warn(
```

Problem 2.1

Compute and print some useful statistics about the data -- the minimum, maximum, mean, and variance of each of the features.

In [74]:
```python
# Compute and print the statistics for each feature
min_values = np.min(X, axis=0)
max_values = np.max(X, axis=0)
mean_values = np.mean(X, axis=0)
variance_values = np.var(X, axis=0)

print("Minimum values for each feature:", min_values)
print("Maximum values for each feature:", max_values)
print("Mean values for each feature:", mean_values)
print("Variance for each feature:", variance_values)
```

```
Minimum values for each feature: [ 197.      190.      214.97    205.42     10.        0.        0.
    0.        1.2189    0.        0.        0.        1.0271 -999.9   ]
Maximum values for each feature: [2.5300e+02 2.4800e+02 2.5202e+02 2.5202e+02 1.7130e+04 1.2338e+04
 9.2380e+03 2.7419e+01 1.8107e+01 1.1368e+01 1.8771e+01 1.4745e+01
 2.7871e+02 7.6920e+02]
Mean values for each feature: [2.41898974e+02 2.28381307e+02 2.41905935e+02 2.33825377e+02
 2.84904650e+03 8.62861100e+02 1.63652650e+02 3.05575493e+00
 6.31144194e+00 1.89391480e+00 4.28955135e+00 2.79775083e+00
 1.04525366e+01 7.65813000e+00]
Variance for each feature: [8.11988160e+01 8.91502653e+01 3.45577443e+01 9.45072114e+01
 1.05055883e+07 3.09041521e+06 6.98073356e+05 7.27689095e+00
 6.18300320e+00 4.15093181e+00 3.94461529e+00 1.93234397e+00
 1.70001843e+02 1.52894736e+03]
```

In past assignments, we have first normalized the data (subtracting the mean and scaling each feature). We will not do that here, however. Why is that step unnecessary for our decision tree model?

Since decision trees make splits based on order and threshold comparisons, the scale or range of features does not influence the splitting decisions. Whether the features are in the range 0–1 or 0–1000, the decision-making process remains the same because the splits only depend on where the data points lie relative to each other within each feature.

Problem 2.2

To allow faster experimentation, select the first 10,000 data points as training data, `Xtr, Ytr`, and the next 10,000 data points as validation, `Xva, Yva`. Then, learn a decision tree classifier on `Xtr` using the `DecisionTreeClassifier` class from `scikit-learn`. Use a large depth, say `max_depth=100`, and the Shannon entropy impurity function in your training. Also, use `random_state=seed` to ensure consistency.

Compute and print out the training error rate and validation error rate of your model.

In [81]:
```
from sklearn.metrics import accuracy_score

# Select the first 10,000 data points as training and the next 10,000 as validation
Xtr, Ytr = data[:10000, :-1], data[:10000, -1]
Xva, Yva = data[10000:20000, :-1], data[10000:20000, -1]


# Create and train the decision tree classifier
clf = DecisionTreeClassifier(max_depth=100, criterion="entropy", random_state=seed)
clf.fit(Xtr, Ytr)

# Calculate training and validation predictions
Ytr_pred = clf.predict(Xtr)
Yva_pred = clf.predict(Xva)

# Calculate error rates
training_error_rate = 1 - accuracy_score(Ytr, Ytr_pred)
validation_error_rate = 1 - accuracy_score(Yva, Yva_pred)

print(f"Training Error Rate: {training_error_rate:.4f}")
print(f"Validation Error Rate: {validation_error_rate:.4f}")
```

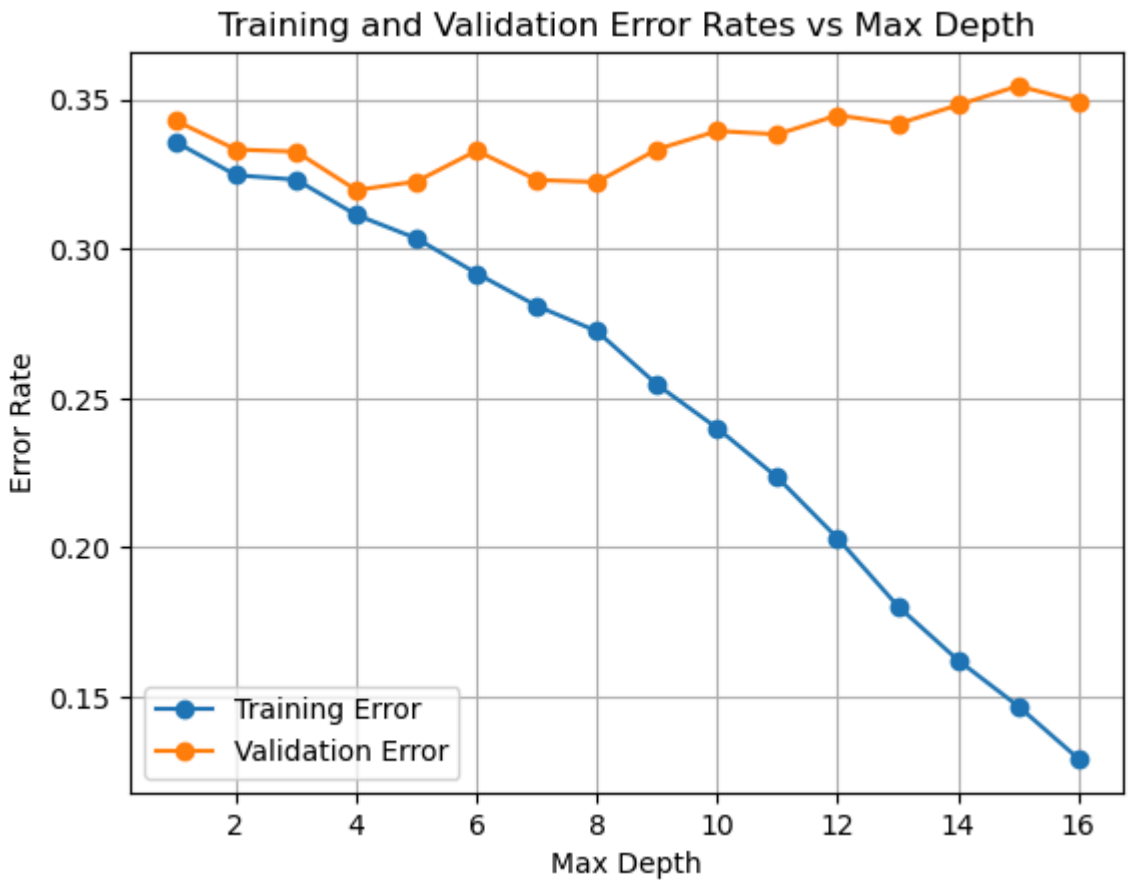
```
Training Error Rate: 0.0098
Validation Error Rate: 0.3823
```

Problem 2.3

Now try varying the `max_depth` parameter, , which forces the tree learning algorithm to stop after at most that many levels. Test `max_depth` values in the range {1, 2, 3, ..., 16}, and plot the training and validation error rates versus `max_depth`. Do models with higher `max_depth` have higher or lower complexity? What choice of `max_depth` provides the best decision tree model?

In [102…

```
# Initialize lists to store error rates
train_errors = []
val_errors = []
depth_list = [i for i in range(1,17)]
# Test max_depth values from 1 to 16
for depth in depth_list:
    # Train a Decision Tree with the specified max_depth
    clf = DecisionTreeClassifier(max_depth=depth, criterion="entropy", random_state=seed)
    clf.fit(Xtr, Ytr)

    # Compute training and validation error rates
    train_error = 1 - accuracy_score(Ytr, clf.predict(Xtr))
    val_error = 1 - accuracy_score(Yva, clf.predict(Xva))

    # Append errors to lists
    train_errors.append(train_error)
    val_errors.append(val_error)

# Plot the training and validation error rates vs max_depth
plt.figure()
plt.plot(range(1, 17), train_errors, label='Training Error', marker='o')
plt.plot(range(1, 17), val_errors, label='Validation Error', marker='o')
plt.xlabel('Max Depth')
plt.ylabel('Error Rate')
plt.title('Training and Validation Error Rates vs Max Depth')
plt.legend()
plt.grid(True)
plt.show()
print("Minimum Validation Error:",min(val_errors), "Best Max Depth:", depth_list[np.argmin(val_errors)] )
```




```
Minimum Validation Error: 0.3197 Best Max Depth: 4
```

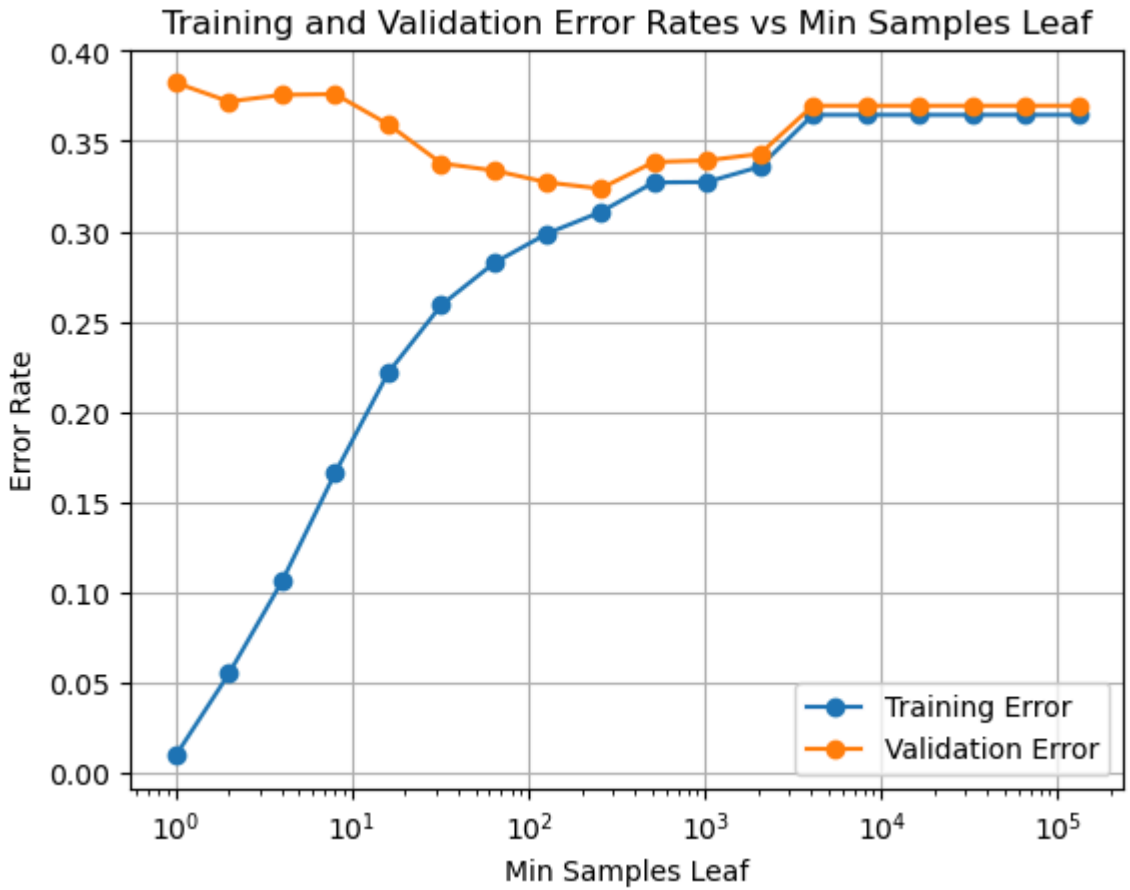
Discuss: "max_depth" = 4 provides the best decision tree model as it has the minimum vadidation error.

Problem 2.4

The `min_samples_leaf` parameter controls the complexity of decision trees by lower bounding the amount of data required to split nodes when learning. Fixing `max_depth=100`, compute and plot the training and validation error rates for `min_samples_leaf` values in the range `{2**0, 2**1, 2**2, ... 2**17}`. Do models with higher `min_samples_leaf` have higher or lower complexity? What choice of `min_samples_leaf` provides the best decision tree model? Is this model better or worse than your depth-controlled model?

In [104…

```python
# Initialize lists to store error rates
train_errors = []
val_errors = []

# Define min_samples_leaf values as powers of 2, from 2^0 to 2^17
min_samples_leaf_values = [2**i for i in range(18)]

# Loop over each min_samples_leaf value
for min_samples_leaf in min_samples_leaf_values:
    # Train a Decision Tree with the specified min_samples_leaf and max_depth=100
    clf = DecisionTreeClassifier(max_depth=100, criterion="entropy", min_samples_leaf=min_samples_leaf, random_stat
    clf.fit(Xtr, Ytr)

    # Compute training and validation error rates
    train_error = 1 - accuracy_score(Ytr, clf.predict(Xtr))
    val_error = 1 - accuracy_score(Yva, clf.predict(Xva))

    # Append errors to lists
    train_errors.append(train_error)
    val_errors.append(val_error)

# Plot the training and validation error rates vs min_samples_leaf
plt.figure()
plt.plot(min_samples_leaf_values, train_errors, label='Training Error', marker='o')
plt.plot(min_samples_leaf_values, val_errors, label='Validation Error', marker='o')
plt.xlabel('Min Samples Leaf')
plt.ylabel('Error Rate')
plt.title('Training and Validation Error Rates vs Min Samples Leaf')
plt.xscale('log')  # Log scale to better visualize the powers of 2
plt.legend()
plt.grid(True)
plt.show()
print("Minimum Validation Error:",min(val_errors), "Best Min Samples Leaf:", min_samples_leaf_values[np.argmin(val_
```




```
Minimum Validation Error: 0.3237 Best Min Samples Leaf: 256
```

Discuss: (1) Higher min_samples_leaf values reduce model complexity. As min_samples_leaf increases, nodes must contain more samples before they can split, resulting in a simpler, more generalized tree.

(2) min_samples_leaf = 256 (2^8) provides the best decision tree model as it has the minimum validation error.

(3) This model is worse than the optimal depth-control model with "max depth" equals to 4 (error = 0.3197). Those two models' performance are almost the same.

Problem 3: Random Forests

Although `scikit` has its own implementations of bagging estimators and random forests, here we will build a random forest model manually, for illustration.

Random Forests are bagged collections of decision trees, which select their decision nodes from randomly chosen subsets of the possible features (rather than all features). You can implement this easily in `DecisionTreeClassifier` using option `max_features=n`, where `n` is the number of features to select from. Since you have ~ 14 features, you might choose, say, `n=10`; smaller values of `n` will make each tree more random, but may require you to have a high maximum depth to ensure you still fit the data effectively. You'll write a for-loop to build the ensemble members, and another to compute the prediction of the ensemble.

Problem 3.1

Using your previous split of the data into training & validation sets, learn a bagged ensemble of decision trees on the training data; then in the next part, you will evaluate the validation performance of the ensemble. (See the pseudocode from lecture slides.) For your individual learners, use parameters that will not constrain the complexity of the resulting tree very much (e.g., set your depth cutoff >= 15, min leaf 1-4, etc.), since the bagging will be used to control overfitting instead.

You will implement bootstrap sampling yourself (again, see pseudocode); simply generate m' data indices uniformly with replacement, using `numpy` functions or simply `rand` and conversion to integers. For your bootstrap process, draw the same number of data as in your training set after the validation split (i.e., set $m' = m$, the number of training data).

Learn at least 50 ensemble members, as you will use them in the next part.

In [117…

```
# Set parameters for the bagging ensemble
num_trees = 50        # Number of ensemble members
max_depth = 15        # Max depth for individual trees
min_samples_leaf = 2  # Min samples per leaf for individual trees

# Seed for reproducibility
```

```
seed = 42
np.random.seed(seed)

# List to hold trained decision tree models
ensemble = []

# Bagging loop to create each individual tree
for i in range(num_trees):
    # Generate bootstrap sample by sampling indices with replacement
    indices = np.random.choice(len(Xtr), size=len(Xtr), replace=True)
    X_bootstrap, Y_bootstrap = Xtr[indices], Ytr[indices]

    # Initialize and train a Decision Tree on the bootstrap sample
    tree = DecisionTreeClassifier(max_features = 10, max_depth=max_depth, min_samples_leaf=min_samples_leaf, random
    tree.fit(X_bootstrap, Y_bootstrap)

    # Append the trained tree to the ensemble
    ensemble.append(tree)

# Display how many models are in the ensemble
print(f"Ensemble created with {len(ensemble)} trees.")
```

```
Ensemble created with 50 trees.
```

Problem 3.2

Plot the training and validation error of the ensemble as a function of the number of learners B that you include in the ensemble, for (at least) values $B \in \{1, 5, 10, 25, 50\}$. (Just use the first B learners that you trained in part 1.)

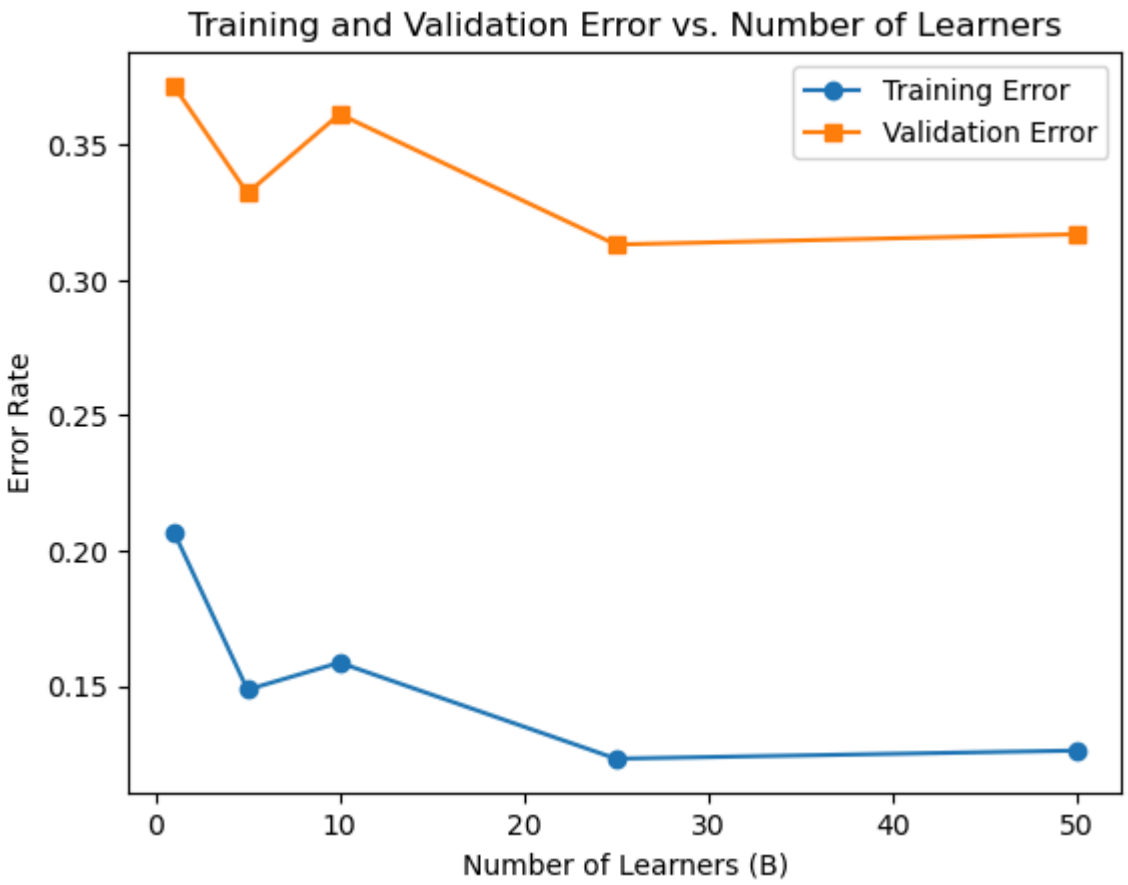
Remember that your ensemble predicts by making a prediction for each member model, and then taking a vote among the B models. Since your target classes are +1 and -1, as in the slide pseudocode, to find the outcome of the vote you can simply check whether the average of the class predictions is positive or negative.

In [120…

```
# Function to calculate ensemble predictions by voting
def ensemble_predict(ensemble, X, B):
    predictions = np.zeros(len(X))
    # Accumulate predictions from the first B models in the ensemble
    for i in range(B):
        predictions += ensemble[i].predict(X)
```

```
    # Average and apply the sign function to get final class (-1 or +1)
    return np.sign(predictions)

# Values of B to test
B_values = [1, 5, 10, 25, 50]
train_errors = []
validation_errors = []

# Calculate training and validation error for each B
for B in B_values:
    # Predict on training data
    Ytr_pred = ensemble_predict(ensemble, Xtr, B)
    train_error = 1 - accuracy_score(Ytr, Ytr_pred)
    train_errors.append(train_error)

    # Predict on validation data
    Yva_pred = ensemble_predict(ensemble, Xva, B)
    validation_error = 1 - accuracy_score(Yva, Yva_pred)
    validation_errors.append(validation_error)

    print(f"B={B}: Training Error={train_error:.4f}, Validation Error={validation_error:.4f}")

# Plotting the results
plt.figure()
plt.plot(B_values, train_errors, marker='o', label="Training Error")
plt.plot(B_values, validation_errors, marker='s', label="Validation Error")
plt.xlabel("Number of Learners (B)")
plt.ylabel("Error Rate")
plt.title("Training and Validation Error vs. Number of Learners")
plt.legend()
plt.show()
```

```
B=1: Training Error=0.2069, Validation Error=0.3717
B=5: Training Error=0.1486, Validation Error=0.3321
B=10: Training Error=0.1586, Validation Error=0.3613
B=25: Training Error=0.1231, Validation Error=0.3131
B=50: Training Error=0.1261, Validation Error=0.3169
```



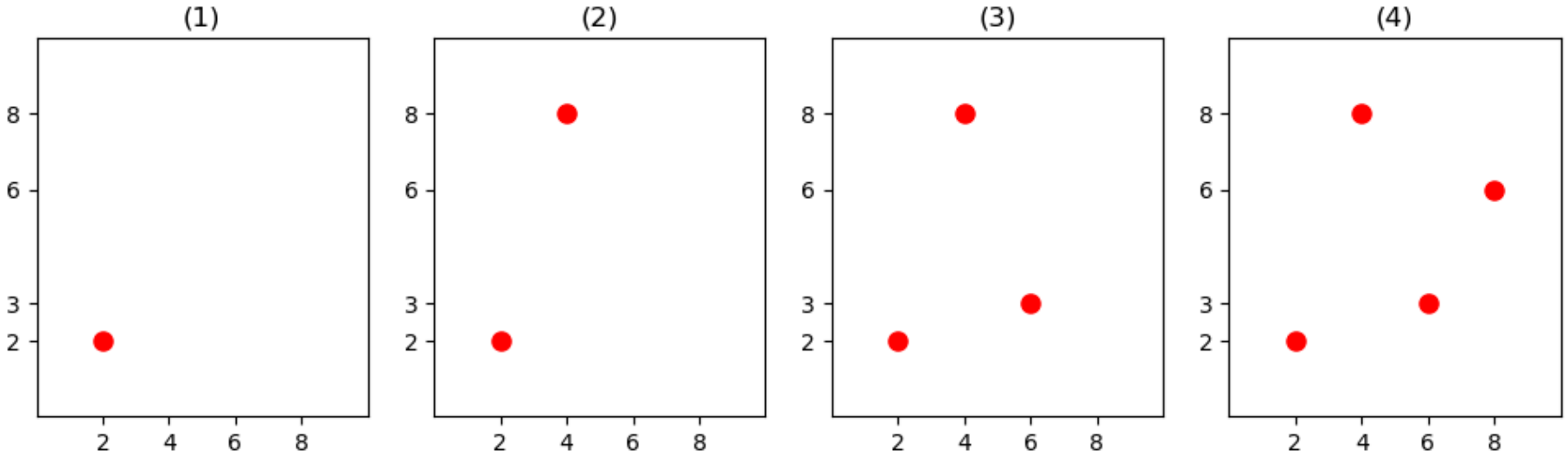
Problem 4: Shattering & VC Dimension

Consider the following four "datasets", which have two real-valued features x_1, x_2:

In [125…

```
fig, axes = plt.subplots(1, 4, figsize=(12, 3))
x = np.array([[2,2],[4,8],[6,3],[8,6]])
for i in range(4):
```

```
    axes[i].plot(x[:i+1,0],x[:i+1,1],'ro',ms=8); axes[i].axis([0,10,0,10]); axes[i].set_title(f'({i+1})')
    axes[i].set_xticks(np.unique(x[:,0])); axes[i].set_yticks(np.unique(x[:,1]));
```



(Note that no three of the four points are co-linear.)

For each of the learners listed below, answer **(a)** which of the four data sets **can be shattered** by a learner of that form? Give a brief explanation / justification (1-2 sentences). Then use your results to **(b)** guess the VC dimension of the classifier (you do not have to give a formal proof, just your reasoning).

In each learner, the parameters $a, b, c, \ldots$ are real-valued scalars, and $T[z]$ is the sign threshold function, i.e., $T[z] = +1$ for $z \geq 0$ and $T[z] = -1$ for $z < 0$.

- $T(\, a + b\, x_1\,)$

In [130…
```
# (a) which data sets & why?
# (b) VC dimension guess?
```

Ans: The decision boundary of this learner is only related to x_1, which is a vertical line dividing the 2D space into left and right parts.

(a) Dataset 1 and dataset 2 can be shattered. For dataset 1, the line can be put in the left / right side of the point when the point is 1 / -1. For dataset 2, the line can be put in the left, right of the 2 points when the points are 1,1 / -1,-1. The line can be put in the the middle of the 2 points when the points are 1,-1 / -1,1.

(b) The VC dimension is 2 as it cannot shatter the dataset with 3 points, when the class of 3 points from left to right are 1, -1, 1/-1, 1, -1

- $T(\ (a * b)\ x_1 + (c/a)\ x_2\ )$

In [ ]:
```
# (a) which data sets & why?
# (b) VC dimension guess?
```

This classifier is linear in both x_1 and x_2, thus it represents a linear function in a 2D space.

(a) Dataset 1, 2, 3 can be shattered, but dataset 4 cannot be shattered when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

(b) The VC dimention is 3. In two-dimensional space, any set of three points can be divided by a single line if they are not collinear (i.e. not on the same line). However, it cannot shatter any datasets with 4 points, when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

- $T(\ (x_1 - a)^2 + (x_2 - b)^2 - c\ )$

In [ ]:
```
# (a) which data sets & why?
# (b) VC dimension guess?
```

This classifier actually divides data in two-dimensional space through a circle with a center at (a, b) and a radius of root c. It can classify data points based on whether they are located inside or outside the circle.

(a) Dataset 1, 2, 3 can be shattered, but dataset 4 cannot be shattered when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

(b) The VC dimention is 3. In two-dimensional space, any set of three points can be divided by a segment of arc. However, it cannot shatter any datasets with 4 points, when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

Extra Credit:

- $T(\ a + b\,x_1 + c\,x_2\ ) \times T(\ d + b\,x_1 + c\,x_2\ )$
  - **Hint:** the two linear equations correspond to two parallel lines, since both have the same coefficients for x_1 and x_2. Then, $T(z) \times T(z') = +1$ if and only if z and z' have the same sign.

In [ ]:
```
# (a) which data sets & why?
# (b) VC dimension guess?
```

(a) Dataset 1, 2, 3, 4 can all be shattered. Since it corrspond to two parallel lines, it can seperate the dataset 4 when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

(b) The VC dimention is 4. As mentioned above, it can shatter any datasets with 4 points, even when the class of 4 points from left to right are 1,-1,-1,1 or -1,1,1,-1.

Statement of Collaboration (5 points)

It is **mandatory** to include a Statement of Collaboration in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using EdD) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content before they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to EdD, etc.). Especially after you have started working on the assignment, try to restrict the discussion to EdD as much as possible, so that there is no doubt as to the extent of your collaboration.

This assignment is completed independently by me.