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 [1]:
```
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
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.

In [2]:
```
```

Problem 1.2 (5 points)

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

In [2]:
```
```

Problem 1.3 (5 points)

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

In [2]:

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 (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 [ ]:
```
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]
```

Problem 2.1

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

In [ ]:

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?

In [ ]:

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 [ ]:

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 [ ]:

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 [ ]:

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 [ ]:

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.)

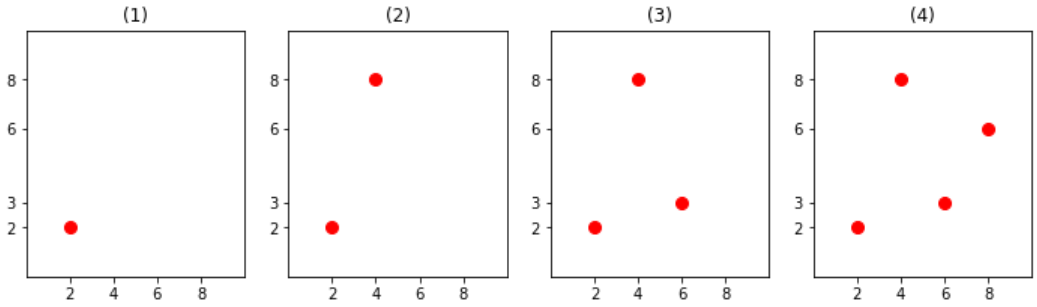
Remember that your ensemble predicts by making a prediction for each member model, and then taking a vote among the B models. Since your target classes are +1 and -1, as in the slide pseudocode, to find the outcome of the vote you can simply check whether the average of the class predictions is positive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

In [ ]:

Problem 4: Shattering & VC Dimension

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

In [16]:
```
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 [ ]:
```
# (a) which data sets & why?
# (b) VC dimension guess?
```

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

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

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

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

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?
```

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.

In [ ]: