CS 5350/6350, DS 4350: Machine Learning Spring 2024

Homework 1

Handed out: January 16, 2024
Due date: January 30, 2024

1 Decision Trees

1. [25 points] Mark loves mangoes. Unfortunately he is lousy at picking ripe mangoes at the grocery. He needs your help. You need to build a decision tree that will help Mark decide if a mango is ripe or not. You need to make the decision based on four features described below:

   (a) **Variety** (*Alphonso, Keitt or Haden*): Describes the variety of the mango.

   (b) **Color** (*Red, Yellow or Green*): Describes the color of the mango.

   (c) **Smell** (*Sweet or None*): Describes the smell of the mango.

   (d) **Time** (*One or Two*): Number of weeks since the mango was plucked.

   You are given the following dataset which contains data for 8 different mangoes. For each mango, the values of the above four features are listed. The label of whether the mango was ripe or not is also provided.

Variety	Color	Smell	Time	Ripe?
Alphonso	Red	None	Two	False
Keitt	Red	None	One	True
Alphonso	Yellow	Sweet	Two	True
Keitt	Green	None	Two	False
Haden	Green	Sweet	One	True
Alphonso	Yellow	None	Two	False
Keitt	Yellow	Sweet	One	False
Alphonso	Red	Sweet	Two	True

Table 1: Training data for the mango prediction problem.

   (a) [5 points] How many possible functions are there to map these four features to a boolean decision? How many functions are consistent with the given training dataset?

for **Variety** there are 3 possible values
For **Color** there are 3 possible values
For **Smell** there are 2 possible values
For **Time** there are 2 possible values
So, the total number of outputs is $3 \times 3 \times 2 \times 2 = 36$
There are 2^16 possible binary ways to fill up those 36 outputs, so the total number of possible functions is $2^{36} = 68719476736$
Since there are 8 examples in the training data. This divides the combinations by $2^8 = 256$, so the number of consistent functions is $2^{36-8} = 2^{28} = 268435456$.

(b) [3 points] What is the entropy of the labels in this data? When calculating entropy, the base of the logarithm should be base 2.

$$Entropy(S) = H(S) = -p_+ log_2(p_+) - p_- log_2(p_-) \quad (1)$$

$H(Ripe) = -\left(\frac{4}{8}\log_2\left(\frac{4}{8}\right) + \frac{4}{8}\log_2\left(\frac{4}{8}\right)\right) = 1$
Essentially, since there is a 50-50 chance of a mango being ripe according to the training data, the entropy is 1.

(c) [4 points] Compute the information gain of each feature and enter it into Table 2. Specify upto 3 decimal places.

Feature	Information Gain
Variety	
Color	
Smell	
Time	

Table 2: Information gain for each feature.

(d) [1 points] Which attribute will you use to construct the root of the tree using the information gain heuristic of the ID3 algorithm?

(e) [8 points] Using the root that you selected in the previous question, construct a decision tree that represents the data. You do not have to use the ID3 algorithm here, you can show any tree with the chosen root.

(f) [4 points] Suppose you are given three more examples, listed in Table 3. Use your decision tree to predict the label for each example. Also report the accuracy of the classifier that you have learned.

Variety	Color	Smell	Time	Ripe?
Alphonso	Green	Sweet	Two	True
Keitt	Red	Sweet	One	False
Haden	Yellow	None	Two	True

Table 3: Test data for mango prediction problem

2. [10 points] Recall that in the ID3 algorithm, we want to identify the best attribute that splits the examples that are relatively pure in one label. Aside from entropy, which we saw in class and you used in the previous question, there are other methods to measure impurity.

   We will now develop a variant of the ID3 algorithm that does not use entropy. If, at some node, we stopped growing the tree and assign the most common label of the remaining examples at that node, then the empirical error on the training set at that node will be

   $$MajorityError = 1 - \max_i p_i$$

   where, p_i is the fraction of examples that are labeled with the i^{th} label.

   (a) [2 points] Notice that $MajorityError$ can be thought of as a measure of impurity just like entropy. Just like we used entropy to define information gain, we can define a new version of information gain that uses $MajorityError$ in place of entropy. Write down an expression that defines a new version of information gain that uses $MajorityError$ in place of entropy.

   (b) [6 points] Calculate the value of your newly defined information gain from the previous question for the four features in the mango dataset from 1. Use 3 significant digits. Enter the information gain into Table 4.

Feature	Information Gain (using majority error)
Variety	
Color	
Smell	
Time	

Table 4: Information gain for each feature.

   (c) [2 points] According to your results in the last question, which attribute should be the root for the decision tree? Do these two measures (entropy and majority error) lead to the same tree?

2 Experiments

This problem uses a modified version of the famous the Mushroom Dataset from the UCI machine learning repository. Each data point has 21 features indicating different characteristics of a mushroom. You can find definitions of each feature in the file `information.txt`. The labels are either `p` or `e`, which stand for poisonous and edible respectively.

Your task is to implement the ID3 algorithm and build a decision tree using the training data `data/train.csv`. The goal of the decision tree is to distinguish between poisonous and edible mushrooms. We also provide the test set `data/test.csv` to evaluate how your decision tree classifier is performing.

Programming notes. You may use any programming language for your implementation, but as discussed in class, we prefer Python. The graders should be able to execute your code on the CADE machines without having to install any libraries. Remember that you are not allowed to use any machine learning libraries (ex: `scikit-learn`, `tensorflow`, `pytorch`, etc.), but can use libraries with mathematical functions like `numpy` and data management libraries like `pandas`.

We have provided a file `data.py`, which is code that can help you with the data loading and manipulation if you choose to use Python. We have included a Jupyter notebook `datademo.ipynb` that demonstrates how to use the file and what functionality it provides. Using the code is not required, although we think it should help you with the implementation of the algorithm. (If you are familiar with a library such as `pandas`, you may find that to be helpful instead of using the provided code.)

Cross-Validation

The depth of the tree is a *hyper-parameter* to the decision tree algorithm that helps reduce overfitting. By depth, we refer to the maximum path length from the root to any leaf. That is, a tree with just a single node has depth 0, a tree with a root attribute directly leading to labels in one step has depth 1 and so on. You will see later in the semester that many machine learning algorithm (SVM, logistic-regression, etc.) require choosing hyper-parameters before training commences, and this choice can make a big difference in the performance of the learners. One way to determine a good hyper-parameter values to use a technique called *cross-validation*.

As usual, we have a training set and a test set. Our goal is to discover good hyperparameters using the training set *only*. Suppose we have a hyperparameter (e.g. the depth of a decision tree) and we wish to ascertain whether it is a good choice or not. To do so, we can set aside some of the training data into a subset called the *validation* set and train on the rest of the training data. When training is finished, we can test the resulting classifier on the validation data. This allows us to get an idea of how well the particular choice of hyper-parameters does.

However, since we did not train on the whole dataset, we may have introduced a statistical bias in the classifier caused by the choice of the validation set. To correct for this, we will need to repeat this process multiple times for different choices of the validation set. That is, we need train many classifiers with different subsets of the training data removed and average out the accuracy across these trials.

For problems with small data sets, a popular method is the leave-one-out approach. For each example, a classifier is trained on the rest of the data and the chosen example is then evaluated. The performance of the classifier is the average accuracy on all the examples. The downside to this method is for a data set with n examples we must train n different classifiers. Of course, this is not practical in general, so we will hold out subsets of the data many times instead.

Specifically, for this problem, you should implement k-fold cross validation.

The general approach for k-fold cross validation is the following: Suppose we want to evaluate how good a particular hyper-parameter is. We randomly split the training data into k equal sized parts. Now, we will train the model on all but one part with the chosen

hyper-parameter and evaluate the trained model on the remaining part. We should repeat this k times, choosing a different part for evaluation each time. This will give we k values of accuracy. Their average cross-validation accuracy gives we an idea of how good this choice of the hyper-parameter is. To find the best value of the hyper-parameter, we will need to repeat this procedure for different choices of the hyper-parameter. Once we find the best value of the hyper-parameter, we can use the value to retrain we classifier using the entire training set.

Growing decision trees

For this problem, your should use the data in `data` folder. This folder contains two files: `train.csv` and `test.csv`. You should train your algorithm on the training file. Remember that you should not look at or use your testing file until your training is complete.

1. [6 points] **Baseline**

   First, find the most common label in the training data. What is the training and test accuracy of a classifier that always predicts this label?

2. **Implementation: Full trees**

   In the first set of decision tree experiments, run the ID3 algorithm we saw in class without any depth restrictions. (That is, there are no hyperparameters for this setting.)

   [6 points] Implement the decision tree data structure and the ID3 algorithm for your decision tree (Remember that the decision tree need not be a binary tree!). For debugging your implementation, you can use the previous toy examples like the toy data from Table 1. Discuss what approaches and design choices you had to make for your implementation and what data structures you used. Also explain the organization of your code.

   Your report should include the following information

   (a) [2 points] The root feature that is selected by your algorithm

   (b) [2 point] Information gain for the root feature

   (c) [2 points] Maximum depth of the tree that your implementation gives

   (d) [3 points] Accuracy on the training set

   (e) [5 points] Accuracy on the test set

3. **Limiting Depth**

   Next, you will perform 5-fold cross-validation to limit the depth of your decision tree, effectively pruning the tree to avoid overfitting. We have already randomly split the training data into five splits. You should use the 5 cross-validation files for this section, titled `data/CVfolds/foldX.csv` where `X` is a number between 1 and 5 (inclusive).

(a) [15 points] Run 5-fold cross-validation using the specified files. Experiment with depths in the set $\{1, 2, 3, 4, 5, 10, 15\}$, reporting the average cross-validation accuracy and standard deviation for each depth. Explicitly specify which depth should be chosen as the best, and explain why. If a certain depth is not feasible for any reason, your report should explain why.

(b) [4 points] Explain briefly how you implemented the depth limit functionality in your code.

(c) [15 points] Using the depth with the greatest cross-validation accuracy from your experiments: train your decision tree on the `data/train.csv` file. Report the accuracy of your decision tree on the `data/test.csv` file.

(d) [5 points] Discuss the performance of the depth limited tree as compared to the full decision tree. Do you think limiting depth is a good idea? Why?

Experiment Submission Guidelines

1. The report should detail your experiments. For each step, explain in no more than a paragraph or so how your implementation works. You may provide the results for the final step as a table or a graph.

2. *Your code should run on the CADE machines.* You should include a shell script, **run.sh**, that will execute your code in the CADE environment. For each experiment, your code must print your results in the following order:

   (a) Entropy of the data

   (b) Best feature and its information gain

   (c) Cross-validation accuracies for each fold (for limiting depth setting)

   (d) Best depth (for the limiting depth setting)

   (e) Accuracy of the trained classifier on the training set (For the limiting depth setting, this would be for the tree with the best depth)

   (f) Accuracy of the trained classifier on the test set (For the limiting depth setting, this would be for the tree with the best depth)

   Without these details, you will lose points for your implementation.

   You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.

3. Please do *not* hand in binary files! We will not grade binary submissions.

3 CS 6350 only: Decision Trees with Attribute Costs

[10 points] Sometimes, we may encounter situations where the features in our learning problem are associated with costs. For example, if we are building a classifier in a medical

scenario, features may correspond to the results of different tests that are performed on a patient. Some tests may be inexpensive (or inflict no harm), such as measuring the patient's body temperature or weight. Some other tests may be expensive (or may cause discomfort to the patient), such as blood tests or radiographs.

In this question, we will explore the problem of learning decision trees in such a scenario. We prefer decision trees that use features associated with low costs at the top of the tree and only use higher cost features if needed at the bottom of the trees. In order to impose this preference, we can modify the information gain heuristic that selects attributes at the root of a tree to penalize costly attributes.

In this question, we will explore different such variations. Suppose we denote $Gain(S, A)$ as the information gain of an attribute A for a dataset S (using the original version of information from ID3). Let $Cost(A)$ denote the cost of the attribute A. We can define two cost-sensitive information gain criteria for attributes as:

1. $Gain_T(S, A) = \frac{Gain(S,A)^2}{Cost(A)}$
2. $Gain_N(S, A) = \frac{2^{Gain(S,A)}-1}{\sqrt{Cost(A)+1}}$

In both cases, note that attributes with higher costs are penalized and so will get chosen only if the information gain is really high.

To evaluate these two methods for root selection, we will use the following training set:

Shape	Color	Size	Material	Label
square	red	big	metal	+
square	blue	small	plastic	+
triangle	yellow	medium	metal	+
triangle	pink	big	leather	-
square	pink	medium	leather	-
circle	red	small	plastic	-
circle	blue	small	metal	-
ellipse	yellow	small	plastic	-
ellipse	blue	big	leather	+
ellipse	pink	medium	wood	+
circle	blue	big	wood	+
triangle	blue	medium	plastic	+

Suppose we know the following costs of the attributes:

Attribute	Cost
Shape	10
Color	30
Size	50
Material	100

1. [8 points] Compute the modified gains $Gain_T$ and $Gain_S$ for each attribute using these costs. Fill in your results in the table below. (upto 3 decimal places)

Attribute	$Gain_T$	$Gain_S$
Shape		
Color		
Size		
Material		

2. [2 points] For each variant of gain, which feature would you choose as the root?

(You may modify your code from the experiments to calculate these values instead of doing so by hand.)