Assignment 5 Project

B351 / Q351

Due: October 29th, 2018 @ 11:59PM

1 Summary

- Gain a basic understanding of machine learning
- Implement entropy and information gain methods for a decision tree algorithm, in addition to recursively building a decision tree from an input node.

To run your program, you may find that you need to install the `numpy` module. You will do this utilizing Pip and using your terminal to execute the command `pip install numpy` (if you're using python3, use `pip3 install numpy` instead). If your code still does not run, please post on piazza or come by office hours.

Please submit your completed files to your private GitHub repository for this class, and upload them to Autolab.

This assignment must be completed **individually**. You must submit your own files to your own repository. Your files will be run against Moss to check for plagiarism within the class.

2 Background

Decision Tree Learning is a machine learning algorithm often used for classification and regression problems. Decision trees offer several advantages over other algorithms, such as ease of implementation and reasonable interpretability, while facing several drawbacks such as a tendency towards overfitting. Our reasoning behind choosing decision trees as the subject for this assignment is that they require minimal mathematical background as compared to other machine learning methods (Linear Regression, Neural Networks, etc). In this assignment, we hope to provide you with some general background knowledge regarding machine learning, as well as give you a chance to work with a real, commonly used machine learning algorithm.

In order to discuss decision trees, it will help to have a common background on machine learning terminology. The following videos provide some background on machine learning, and I highly recommend you watch them. (Having this baseline of machine learning knowledge is a prerequisite for a good machine learning related final project!)

1. What is Machine Learning? (7:15)
2. Introduction to Supervised Learning (12:29)

Decision trees can be used for both classification and regression problems, but for this assignment we will be limiting our scope to classification. Even further, we limit the type of data which our algorithm can receive to categorical data. If you would like to try some of your own datasets with this decision tree implementation, you must ensure that your data is categorical, and the attributes (not the labels) are one-hot encoded. If you aren't sure what this means, try searching it or don't worry about it.

How does a decision tree work? There are two main phases: 1. learn the tree and 2. classify data.

2.1 Learning the Tree

Learning the tree is a recursive process. We start with a root node, which contains all data points. We will use some criterion to decide which attribute to use to split the data into 2 subnodes. Repeat this process at each of the subnodes, using the data they receive from their parent node. The process continues until a node is reached, wherein all the data is of the same class, or has identical attributes. Your job is to implement that criterion. For implementation details of the algorithm itself, refer to the `DecisionTreeFactory` class in `decision_tree_factory.py`.

In the textbook, there are detailed explanations on how to build a decision tree. For further information, please refer to section 8.4 (page 184) in the book. (You can find the book in syllabus.)

2.2 Classify Data

Once the decision tree is built, classifying new datapoints is trivial (and already implemented for you). Simply propagate that datapoint through the tree, and when you arrive at a leaf node, return the class of the most common data in that node.

3 Programming Component

3.1 Data Structures

The following two classes are the primary classes where you will be writing code. See the Objective section for what exactly you need to implement.

3.1.1 DTreeNode

This class is in the `node.py` file and contains functions for the nodes in a decision tree. A `DTreeNode` (as opposed to a `Node` seen in this file) does all the things a Node does and has all the same attributes, in addition to keeping track of indexes. It has one additional attribute:

- **idxs** - an array of indexes for the node

The following are DTreeNode's object methods:

1. **get_idxs(self)** - returns an array of indexes (you can consider them as data points)
2. **get_entropy(self, labels)** - returns the entropy of the node. As explained later, it is your responsibility to implement this function.
    - **labels** - an array of the labels of the data points at the node, e.g. [1., 2., 0.]
3. **should_stop(self, data)** - returns a boolean that indicates whether or not the decision tree building should stop at the node
    - **data** - a list of the data for the node
4. **set_class(self, unique, counts)** - sets the most likely class of data points in the node.
    - **unique** - an array of sorted unique values
    - **counts** - an array of counts that indicate the number of times each of the unique values comes up in the original array

3.1.2 DecisionTreeFactory

This class is in the `decision_tree_factory.py` file and contains functions to put together a decision tree. It has two attributes:

- **data** - a matrix that contains the training data from which to build a tree
- **labels** - a matrix that contains the labels (classifications) for the training data

Note that data and labels are separate entities, but their indexes correspond with each other. Indexes for the data points are the same indexes for the labels.

The following are DecisionTreeFactory's object methods:

1. **__init__(self, data, labels)** - constructor for the `DecisionTreeFactory` class.

2. **build_tree(self)** - builds a tree from data and labels

3. **_build_tree_rec(self, node)** - recursively builds tree out from the input node.

    - **node** - the DTreeNode from which to continue building out our tree.

4. **_get_ranked_splits(self, splits)** - returns a list of nodes for possible splits ranked by information gain (high to low). You will need to implement this function.

5. **_calc_information_gain(self, parent_node, children_nodes)** - returns the information gain from splitting with the given attributes. As explained later, it is your responsibility to implement this function.

    - **parent_node** - the DTreeNode from which we are generating the split
    - **children_nodes** - a list of DTreeNodes that are the child nodes for this potential split

The next four classes are provided for you. No code of yours is needed in these, but it's useful to know what they're doing.

3.1.3 Node

This class is in the `decision_tree_factory.py` file. `Node` is a node in a decision tree. It keeps track of the attribute split for its children and is used in the creation of a `DTreeNode`. It has four attributes:

- **leaf** - boolean that says if the node is a leaf or not
- **children** - a dictionary of children nodes
- **attr** - a number that represents the attribute of the node
- **predicted** - a number that represents the most likely class of the node

3.1.4 DecisionTree

This class is in the `decision_tree.py` file. It is a wrapper that starts at the root node of a decision tree and classifies the data.

3.1.5 Loader

This class is in the `loader.py` file. `Loader` provides utilities for loading data sets.

3.1.6 Driver

This class is in the `driver.py` file. `Driver` runs and reports details of running the decision tree algorithm on data sets.

3.2 Objective

Your goal is to complete the following tasks (probably in the order they are presented). As stated before, the actual classifying and creating of the decision tree has been implemented for you already, so your job is to just implement the following three functions, which are involved in the decision tree process.

3.2.1 DTreeNode.get_entropy(labels) (40%)

Entropy is a concept from information theory, which in vague terms, describes the amount of information missing from a system. This method should use the information contained in the given labels to determine the entropy value for a node in a decision tree.

3.2.2 DecisionTreeFactory._calc_information_gain(parent_node, children_nodes) (40%)

Information Gain is a metric for choosing decision tree splits, returning a value based on the difference between the entropy of the parent and the normalized entropy of all the children.

To test your implementations, you will need to run your project in `driver.py`. The following is the output that you should expect:

```
Voting dataset error:  0.06481481481481481
 Scale dataset error:  0.1987179487179487
```

3.2.3 DecisionTreeFactory._get_ranked_splits(self, splits) (20%)

Your job is to sort the given list of possible splits in the order of information gain.

3.3 Bonus (up to 30%)

This assignment covers only a tiny portion of decision trees. If you have an interest in exploring the topic further, then we will offer bonus points for students who demonstrate significant additional work in exploring the topic. To receive bonus points, your bonus attempt code must be added to your GitHub repository, and you must submit a neat, well formatted pdf report to the 'Assignment 5 Project' assignment on Canvas. The report should explain, with examples, what you did and why, and how it affected performance compared to other methods on at least 2 datasets (the provided datasets are acceptable). In this report, you must clearly specify how many bonus points you expect to receive for your work.

Some ideas for a bonus implementation include using a different metric for the decision tree (e.g. GINI, misclassification error) (expect 10% bonus), making the algorithm run on continuous attributes (expect 20% bonus), making the algorithm perform both classification and regression, or making a random forest algorithm (expect 30% bonus).