COSC 2753 | Machine Learning

Week 7+1 Lab Exercises: **Rule Learning**

Introduction

In this lab you will be:

1. Implement the entropy calculation
2. Implement a simplified proposition rule learning algorithm, outputting rules

sklearn does not have an implementation of a rule learner. Instead you will implement a simplified CN2 algorithm. This algorithm will construct pre-conditions that contain a single term, that is, the rule precondition will not contain conjunctions. This will require you to implement functions in python, and use simple loops and if-statements. If you are unfamiliar with these, first revise the Python tutorials from Lab01.

This lab only requires Pandas/Numpy to load with work with the data set, and the math library.

```
import pandas as pd
import numpy as np
import math
```

Datasets

You will be looking at two data sets for this lab which you have seen before:

1. Sailing days
2. Zoo (animal) classification

You can download these from Canvas or BitBucket code repo.

```
sailData = pd.read_csv('./Lab/sailing-custom-python.txt',delim_whitespace=True)
zooData = pd.read_csv('./Lab/zoo-python.txt',delim_whitespace=True)
```

```
sailData.head()
```

	Outlook	Company	Sailboat	Sail
0	rainy	big	big	yes
1	rainy	big	small	yes
2	rainy	med	big	no
3	rainy	med	small	no
4	sunny	big	big	yes

```
zooData.head()
```

	hair	feathers	eggs	milk	airborne	aquatic	predator	toothed	backbone	breathes	venomous	fins	legs	tail	domestic	catsize	type	name
0	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes	No	No	4.0	No	No	Yes	mammal	aardvark
1	Yes	No	No	Yes	No	No	No	Yes	Yes	Yes	No	No	4.0	Yes	No	Yes	mammal	antelope
2	No	No	Yes	No	No	Yes	Yes	Yes	Yes	No	No	Yes	0.0	Yes	No	No	fish	bass
3	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes	No	No	4.0	No	No	Yes	mammal	bear
4	Yes	No	No	Yes	No	No	Yes	Yes	Yes	Yes	No	No	4.0	Yes	No	Yes	mammal	boar

remove unnecessary columns:

```
zooData = zooData.drop(columns='name')
```

Simple Rule Learner

You will develop the simple rule learner over three parts:

1. Entropy calculation function
2. Majority class calculation function
3. Rule learner

Entropy function

First you will need a function that calculates the entropy of a data set.

Note: In Juypter you need to place the entire function definition in a single input group. You also need to obey formatting rules for functions (that is tabs/spaces for indentation)

This function takes two parameters, (1) the data set, and (2) the column name of the output/target class. The function should return the entropy of the data set.

As a reminder, entropy is: Entropy is a measure of the randomness in the information being processed.

```
def entropy(data, target):
    #TODO
```

$\text{entropy}(S) = -\sum_{i=0}^{c} p_i \log_2 p_i$

The pseudo-code for the entropy calculation is ($x indicates variable x):

```
entropy($data, $target):
    $entropy_value = 0
    foreach $value of $target:
            $count = the number of examples in $data where $value==$target
            $p_i = $count / (total number of examples in $data)
            Add to $entropy_value using $p_i
        return $entropy_value
```

The following code-snippets will help in creating the entropy function:

- You can get a count of each of the values of a single attribute using:

  ```
  vCounts = pd.value_counts(data[target])
  ```

  This gives as a 2D array, for each value of the target column, the number of values matching that value.
- You can iterate through the actual counts by:

  ```
  for value in vCounts:
  ```
- You can iterate through the labels of the value counts array by:

  ```
  for value in vCounts.axes[0]:
  ```
- The following returns all examples in the data frame whose attribute matches the given value:

  ```
  matching = data.loc[data[attribute] == value]
  ```
- The number of rows in a data frame is

  ```
  data.shape[0]
  ```
- The size property of a pandas data frame returns the number of elements in the data frame, or the length of a single column:

  ```
  data.size
  ```
- The log_2 of a number x is calculated by:

  ```
  math.log(x,2)
  ```

```
def entropy(data, target):
    entropy_value = 0
    data_size = data.shape[0]
    vCounts = data[target].value_counts()
    #print(data_size)
    #print(vCounts)

    for count in vCounts:
            #count = the number of examples in $data where $value==$target
            p_i = count / data_size
            if p_i!=0:
                entropy_value += -p_i*math.log(p_i,2)

    return entropy_value
```

```
print('Entropy for Sail data: ', entropy(sailData, 'Sail'))
print('Entropy for Zoo: ', entropy(zooData, 'type'))
```

```
Entropy for Sail data:  0.9975025463691153
Entropy for Zoo:  2.390559682294039
```

If you have implemented the entropy function correctly, you should get the following results for the sailing and zoo data sets:

```
- entropy(sailData, 'Sail') = 0.9975025463691153
- entropy(zooData, 'type') = 2.390559682294039
```

Majority Class

Secondly, you will need to implement a function that returns the value of the target column which has the majority number of values. This code should be very similar to the entropy calculation. Use the following as the definition for your function:

```
def majority_class(data, target):
    #TODO
```

The pseudo-code for finding the majority is:

```
majority_class($data, $target):
    $majority = 0
    $class = ''
    foreach $value of $target:
        $count = the number of examples in $data where $value==$target
        if $count > $majority:
            $majority = $count
```

```
            $class = $value
    return $class
```

Alternatively, you can investigate how to use the **idmax()** function, which is a function of a pandas dataframe/series.

```
def majority_class(data, target):
    majority = 0
    class_name = ''
    vCounts = data[target].value_counts()
    #print(vCounts)

    for trait in vCounts.axes[0]:
        #count = the number of examples in $data where $value==$target
        matching = data.loc[data[target] == trait]
        count = matching.shape[0]
        #print(trait, count)
        if count > majority:
            majority = count
            class_name = trait
    return class_name
```

Here's a step-by-step explanation of what the function does:

1. The function starts by initializing two variables, `majority` and `class_name`, to 0 and an empty string respectively. These variables will be used to keep track of the class that appears most frequently and the number of times it appears.
2. The `value_counts` method is called on the target column of the DataFrame. This method returns a Series containing counts of unique values in descending order, so the first element is the most frequently-occurring element. The resulting Series is stored in the `vCounts` variable.
3. The function then iterates over the unique values (traits) in the target column, which are the index of the `vCounts` Series. For each trait, it creates a new DataFrame `matching` that only includes the rows from the original DataFrame where the target column is equal to the current trait. The number of rows in this DataFrame (which is the number of times the current trait appears in the target column) is then calculated using the `shape` attribute.
4. If the count of the current trait is greater than the count of the previously most frequent trait (stored in the `majority` variable), the function updates `majority` and `class_name` with the count and the name of the current trait.
5. After all traits have been processed, the function returns the name of the most frequent class.

In summary, this function calculates and returns the most frequent class in the target column of a given DataFrame.

```
print('Majority for Sail data Target: ', majority_class(sailData, 'Sail'))
print('Majority for Zoo data Target: ', majority_class(zooData, 'type'))
```

```
Majority for Sail data Target:  yes
Majority for Zoo data Target:  mammal
```

Rule Learner

Given the above two functions, it is now possible to implement a simple propositional rule learner. The features of this rule learner are:

1. The pre-condition of each rule contains a single condition
2. All attributes are treated as categorical
3. The rules are going to be printed to the command line

The pseudo-code for this simple propositional rule learner is:

```
simpler_rule_learner($data, $target):
    while $data.shape[0] > 0:
        if entropy($data) = 0:
            print ("otherwise =>", majority_class($data,$target))
            drop all rows in $data
        else:
            $best_entropy = entropy($data)
            $best_attribute = ''
            $best_value = ''
            $best_data=$data
            foreach $attribue of $data:
                foreach $value of $attribute:
                    $data2 = select the examples in $data where $attribute==$value
                    if entropy($data2) < $best_entropy:
                        $best_entropy = entropy($data2)
                        $best_attribute = $attribue
                        $best_value = $value
                        $best_data=$data2
            print($best_attribute, "=", $best_value, "=>",
                    majority_class($best_data,$target))
            drop all rows of $data2 from $data
```

```
def simpler_rule_learner(data, target):
    # TODO
```

Hints:

- You can drop all the necessary row of `$data` by constructing the opposition condition that was used to create `$data`, ie

  ```
      data = data.loc[data[best_attribute] != best_value]
  ```

- The following drops all rows of a data frame

  ```
      data = data.iloc[0:0]
  ```

If you have implemented the simple ruler learner correctly, you should get the following output

```
simpler_rule_learner(sailData, 'Sail')
    Company = big => yes
    Outlook = rainy => no
    Sailboat = small => yes
    Company = med => yes
    otherwise => no
```

```
simpler_rule_learner(zooData, 'type')
    feathers = Yes => bird
    milk = Yes => mammal
    fins = Yes => fish
    hair = Yes => insect
    airborne = Yes => insect
    legs = 8.0 => invertebrate
    catsize = Yes => reptile
    eggs = No => reptile
    breathes = No => invertebrate
    aquatic = Yes => amphibian
    tail = Yes => reptile
    legs = 0.0 => invertebrate
    otherwise => insect
```

```
def simpler_rule_learner(data, target):
    while data.shape[0] > 0:
        if entropy(data,target) == 0:
            print ("otherwise =>", majority_class(data,target))
            data = data.iloc[0:0] #drop all rows in $data
        else:
            best_entropy = entropy(data,target)
            best_attribute = ''
            best_value = ''
            best_data=data
            for attribute in data.columns:
                vCounts = data[attribute].value_counts()
                for value in vCounts.axes[0]:  #for value in  $attribute:
                    #data2 = select the examples in $data where $attribute==$value
                    data2 = data.loc[data[attribute] == value]
                    if entropy(data2,target) < best_entropy:
                        best_entropy = entropy(data2,target)
                        best_attribute = attribute
                        best_value = value
                        best_data=data2
            print(best_attribute, "=", best_value, "=>",
                    majority_class(best_data,target))
            data = data.loc[data[best_attribute] != best_value]#drop all rows of $data2 from $data
```

Here's a step-by-step explanation of what the function does:

1. The function starts with a while loop that continues as long as there are rows left in the DataFrame.
2. Inside the loop, the function first checks if the entropy of the target column is 0 using the `entropy` function. If it is, this means all remaining examples belong to the same class, so the function prints a rule that predicts this class for all remaining examples and then drops all rows from the DataFrame.
3. If the entropy is not 0, this means there are still examples from different classes left in the DataFrame. The function then initializes four variables (`best_entropy`, `best_attribute`, `best_value`, and `best_data`) to keep track of the attribute-value pair that results in the lowest entropy and the subset of the DataFrame where this pair is true.
4. The function then iterates over all attributes in the DataFrame and all unique values in each attribute. For each attribute-value pair, it creates a new DataFrame `data2` that only includes the rows where the attribute is equal to the value. It then calculates the entropy of the target column in this DataFrame.
5. If the entropy of `data2` is lower than the current best entropy, the function updates the best entropy, attribute, value, and data with the entropy, attribute, value, and data of `data2`.
6. After all attribute-value pairs have been processed, the function prints a rule that predicts the majority class in `best_data` for examples where `best_attribute` is equal to `best_value`. It then drops all rows from the DataFrame where `best_attribute` is equal to `best_value` and continues with the next iteration of the loop.

In summary, this function learns a set of rules to predict the target column based on the other columns in the DataFrame by iteratively finding the attribute-value pair that results in the lowest entropy and printing a rule based on this pair. The function uses the `entropy` and `majority_class` functions to calculate the entropy of a column and find the majority class in a column, respectively.

```
simpler_rule_learner(sailData, 'Sail')
```

```
Company = big => yes
Outlook = rainy => no
Company = med => yes
Sailboat = small => yes
otherwise => no
```

```
simpler_rule_learner(zooData, 'type')
```

```
feathers = Yes => bird
milk = Yes => mammal
hair = Yes => insect
airborne = Yes => insect
fins = Yes => fish
legs = 8.0 => invertebrate
eggs = No => reptile
breathes = No => invertebrate
aquatic = Yes => amphibian
predator = Yes => reptile
backbone = Yes => reptile
legs = 6.0 => insect
otherwise => invertebrate
```

Sample Solutions

If you are struggling with the first two functions, a sample solution has been provided for these. Only use this if you have **made your absolute best attempts** at implementing these functions yourself. The purpose of this lab is to understand common aspects of symbolic machine learning algorithms, though the CN2 algorithm. You will gain significantly less out of this lab if you don't try to solve the problems yourself.