Machine Learning

Program assignment #1

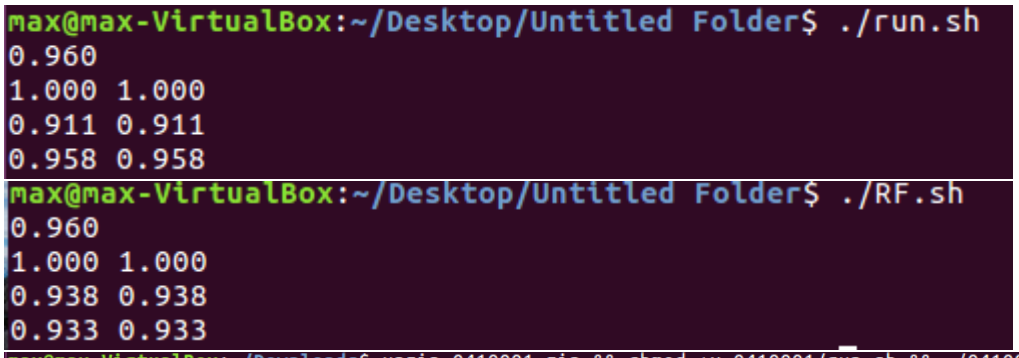
0410001

Hong-Shuo, Chen

1. Problem

   For this assignment, you need to implement ID3 algorithm to construct a decision tree and use K-fold cross validation (K=5) to validate classification performance by outputting precision and recall for each class and total accuracy.

2. Environment
   - Ubuntu 16.04.3 LTS
3. Using library and language
   - Library: numpy, math
   - Language: Python 3.5.2
4. Results



5. Code

   I define the following function:

   - get_attributes(data)

     handle continous descriptive features and get the attributes set

     Parameters:

     - data: numpy.ndarray

     return: attributes : numpy.ndarray

   - entropy(data)

     calculate the entropy

     Parameters:

     - data: numpy.ndarray

     return: entropy: float64

   - rem(data,attr)

     calculate the remainder

     Parameters:

     - data: numpy.ndarray
     - attr: dtype = {'names' = ('attr', 'threshold'),'formats' = ('U20','f8')}

     return: entropy: float64

   - gain(data, attr)

     calculate the information gain

     Parameters:

     - data: numpy.ndarray
     - attr: dtype = {'names' = ('attr', 'threshold'),'formats' = ('U20','f8')}

return: gain: float64

- choose_best_attr(data, attributes)

  choose the best attribute

  Parameters:

  - data: numpy.ndarray
  - attr: numpy.ndarray

  return: best: int32

- majority_value(data)

  get the most class in the dataset

  Parameters:

  - data: numpy.ndarray

  return: class : ‘U20’

- create_decision_tree(data, attributes, parent)
  create decisiontree with attribute as label

  Parameters:

  - data: numpy.ndarray
  - attributes: numpy.ndarray
  - parent: 'U20'

  return: tree: dict

- create_decision_tree_ts(data, attributes, parent)
  create decisiontree with threshold as label

  Parameters:

  - data: numpy.ndarray
  - attributes: numpy.ndarray
  - parent: 'U20'

  return: tree: dict

- K_fold_cross_validation(data,k)
  do the K-fold cross validation and calculate the precision, recall, and accuracy

  Parameters:

  - data: numpy.ndarray
  - k: int

  return:

- random_forest(data,trees)
  construct a multitude of decision trees at training time and output the the result that is the most predictions of the classes of the individual trees

  Parameters:

  - data: numpy.ndarray
  - trees: int

  return:

6. Further explanation of the code

```
#create decisiontree with attribute as label
def create_decision_tree(data, attributes, parent):
    target = data['class']
    # if the dataset is empty return the classification of the parent
    if data.size == 0:
        return parent
    # If the attributes list is empty, return the
    # default value. When checking the attributes list for emptiness, we
    # need to subtract 1 to account for the target attribute.
    elif attributes.size == 0:
        return majority_value(data)
```

```
# If all the records in the dataset have the same classification,
 # return that classification.
elif np.count_nonzero(target == target[0]) == target.size:
        return target[0]
else:
        # Choose the next best attribute to best classify our data
        id = choose_best_attr(data,attributes)
        best = attributes[id]
        tmp = np.delete(attributes,id)
        # Create a new decision tree with the best attribute and an empty
        # dictionary object--we'll fill that up next.
        tree = {best['attr']:{}}
        # Create a new decision sub-node for each of the values in the
        # best attribute field
        data0 = data1 = np.array([],dtype = data.dtype)
        for i in range(data.size):
                if data[i][best['attr']] < best['threshold']:
                        data0 = np.append(data0,data[i])
                else:
                        data1 = np.append(data1,data[i])
        # Add the new subtree to the empty dictionary object in our new
         # tree/node we just created.
        subtree = create_decision_tree(data0, tmp, majority_value(data))
        tree[best['attr']][0] = subtree
        subtree = create_decision_tree(data1, tmp, majority_value(data))
        tree[best['attr']][1] = subtree
return tree
```