Using map-reduce framework to compute AVC-sets for efficient construction of decision trees

Data mining lab 2

Lab outline

- Python data structures
- Map reduce framework
- AVC-sets

Python data structures. Tuples

- An array of elements in Python is called *sequence*.
- If you separate elements of the sequence by comma, you get a *tuple*:
  - (1,2,3) ('a','b') ((1,3),(2,4,6))
- The elements of the tuple cannot be changed – tuples are immutable sequences, you cannot assign new values to the elements of a tuple. Hence, tuples are read-only data structures.
- The keys of the dictionaries in Python must be immutable, so we will use tuples as the keys. The only other data structure which can be used as dictionary key is string, which is also immutable.

Operations on tuples

- Initialize:
  - from scratch: x=1,2,3
  - from an array: x=tuple([1,2,3])
  - from string: y=tuple('abc')
  - by assignment: (z,w)=(4,5)
- Read element:
  - x[1] - prints 2
  - y[2] - prints c
- Read range:
  - x[0:2] - prints (1,2)

Python data structures. Lists

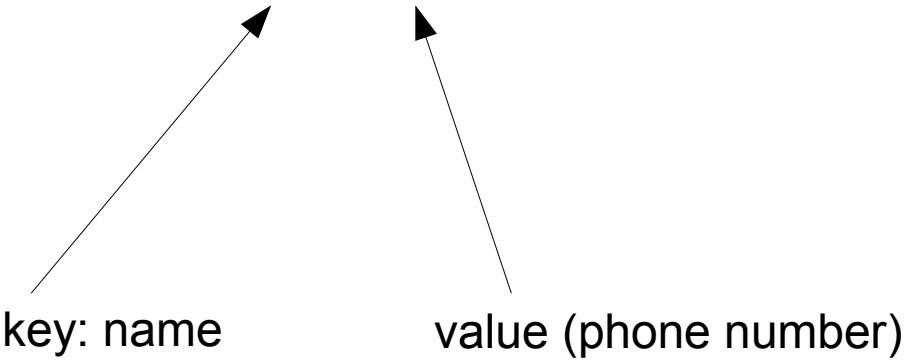
- *Lists* in Python are the variable-size arrays. They can be modified, they are *mutable* sequences, i.e. you can change an element of a list at the specific position.
- To create a list:
  - x=[ ] (list without elements)
  - y=[None, None, None] (an array of 3 NULL elements)
  - z=['bunny',12] (2-elements array, each element is of different type)

Operations on lists

- Read an element of the list (exactly the same as for tuples)
  - numbers=[1,2,3,4,5,6,7,8,9,10]
  - numbers[7:10] (prints [8,9,10])
  - numbers[-3:-1] (prints [8,9], counts from the end)
- Appending new elements
  - a=[1,2,3] b=[4,5,6] print a+b (prints [1,2,3,4,5,6])
  - a.append(4) print a (prints [1,2,3,4])
  - a.insert(1,0) print a (prints[1,0,2,3,4])
- Deleting an element
  - a.remove(2) (prints [1,0,3,4], since removed an element equal to 2)
  - del a[1] (prints [1,3,4], since removed element at position 1)
- Assignment to a specific array position
  - word=list('pearl') (word=['p','e','a','r','l'])
  - word[2:]=list('ar') (word=['p','e','a','r'])

Python data structures. Dictionaries

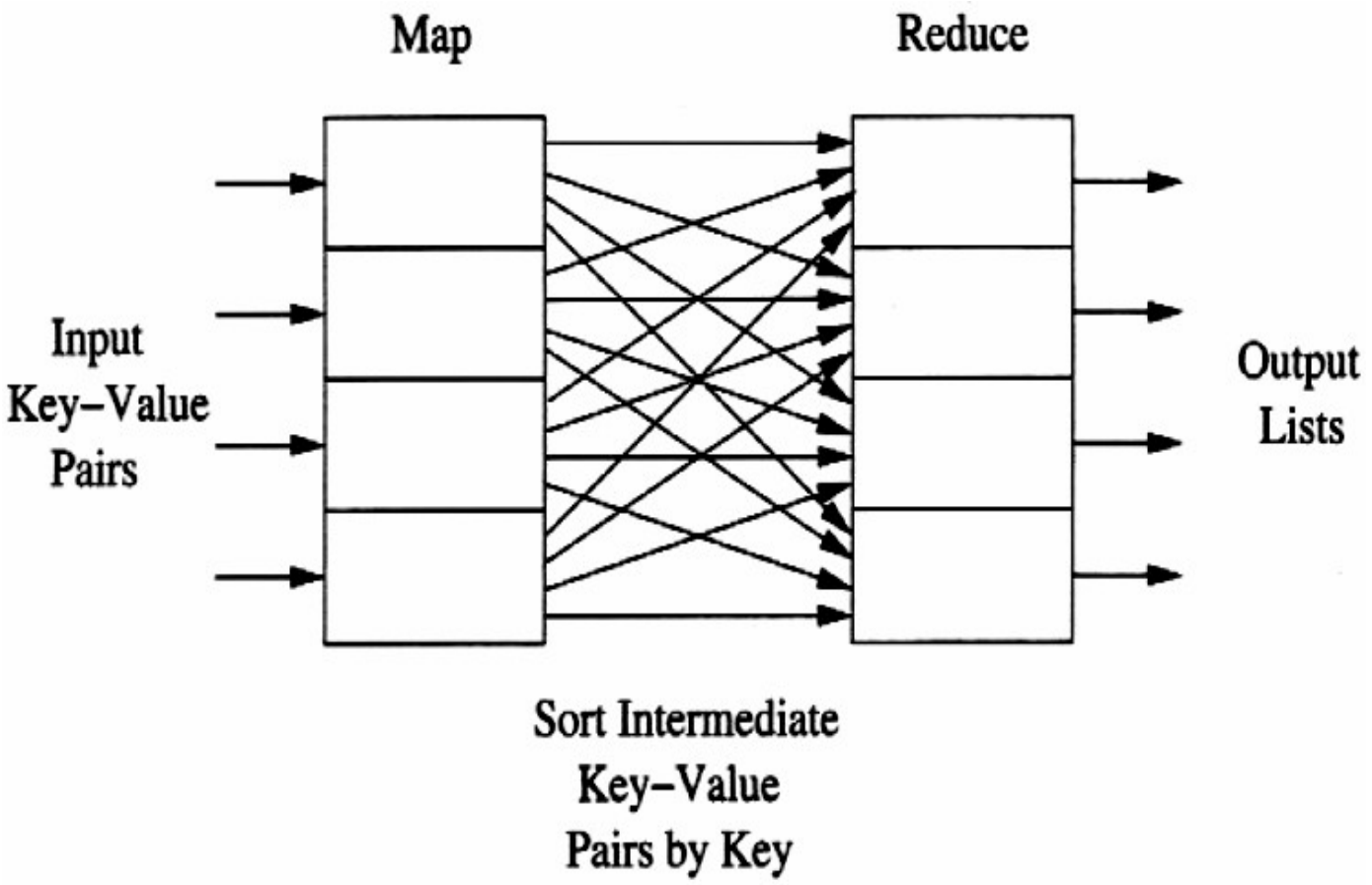
- *Dictionaries* consist of pairs. Each pair is a key-value pair. Each pair is called item.
- To create a dictionary:
  - phonebook={} %(empty dictionary)
  - phonebook={'Mary':1254, 'John': 1321, 'Dick': 1511}



Operations on dictionaries

- Assignments
  - phonebook ['Liz']=3455 (added a new entry)
  - phonebook['Mary']=2211 (replaced an old entry)
- Looking for a key
  - print phonebook.get('Mary') (prints 2211)
  - print phonebook.get('Kate') (prints None)
  - phonebook.has_key('John') (prints 1)
  - name='Liz' print (name in phonebook) (prints 1 - true)
- Iterating through the dictionary
  - phonebook.items()
    - prints {'Mary':1254, 'John': 1321, 'Dick': 1511}
  - it=phonebook.iteritems()
    - Returns an iterator object which can be converted to list and scanned inside the code

MapReduce Framework



Attribute, Value, Class (AVC)-sets

The best split for a node of the decision tree can be determined efficiently if we have the AVC-sets for the node

(AVC stands for **A**ttribute-**V**alue, **C**lass label).

AVC-sets are typically small and hopefully fit the main memory

For example, for a row:

sunny,85,85,FALSE,no

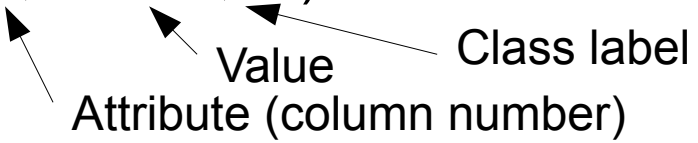
the AVC set with 4 tuples should be generated

(0, 'sunny', no)

(1, 85, no)

(2, 85, no)

(3, 'False',no)



AVC-sets generation in MapReduce framework. Map function

We are going to code AVC-set generation for Map Reduce framework in Disco.

For this we need to implement two functions: *fun_map* and *fun_reduce*

fun_map takes as a parameter one line of the input file (with delimiter), splits it into the list of attributes, and for each attribute except the last (class) generates a tuple (column_index, attribute_value, class_value). For example for a row: sunny,85,85,FALSE,no

4 tuples will be generated as a dictionary keys and inserted into a dictionary with count 1 each:

[(0, 'sunny', no), 1]

[(1, 85, no),1]

[(2, 85, no), 1]

[(3, 'False',no),1]

Look at the code for words count to see how words counts are generated from a line of text.

Modify map_reduce so it produces a dictionary with AVC counts.

The parameter *delimiter* should be passed inside *params* argument which is a Disco object.

So extract it from there by *params.delimiter* (or for a local test just hard code it to be comma delimiter)

AVC-sets generation in MapReduce framework. Reduce function

fun_reduce takes as an argument each element of a dictionary produced by fun_map (in a distributed environment the elements with the same key ends up in the same machine), and aggregates counters for each AVC tuple (see for an example the words count code again).

The file avc_local.py can be used to test you code. For this you need the implemented by Dr. Thomo mapreducelocal.py as well as the sample input weatherdata.txt

Enjoy programming in Python!

MapReduce Framework. Disco

Disco is installed and is running on cluster. The master machine is dbssh1.cs.uvic.ca. You will need a user name and a password in order to login into this machine.

Tutorials on how to run your job on Disco can be read at http://discoproject.org/doc/start/tutorial.html

Follow the following steps

1. Copy your tested fun_map and fun_reduce implementations into file avc_cluster.py. Pay an attention how do we pass additional parameters to map_reduce function. To extract your parameter from params argument do params.paramname.

2. Prepare your input.
    a. Break input file into chunks
        mkdir test2
        split -l 4 weatherdata.txt test2/test2-
        This creates 4 files with 4 lines each in the directory test2
    b. Copy input chunks to cluster nodes
        source opt/etc/disco/disco.conf
        python ./src/disco/util/distrfiles.py test2 ./opt/etc/disco/nodes > test2.chunks
        After you run the Disco script distrfiles.py input chunks end up in 4 cluster nodes,
        and test2.chunks contains the paths of these input files.
3. Run your program
    python avc_cluster.py http://localhost:8989 `cat test2.chunks` > test2.results
        First command line argument is the location of Disco master,
        and the second argument is the list of input files (our input chunks)
4.. Check output file test2.results
5. Repeat the procedure for another bigger input file soybean.txt which represents the training set for classifying soybean diseases.