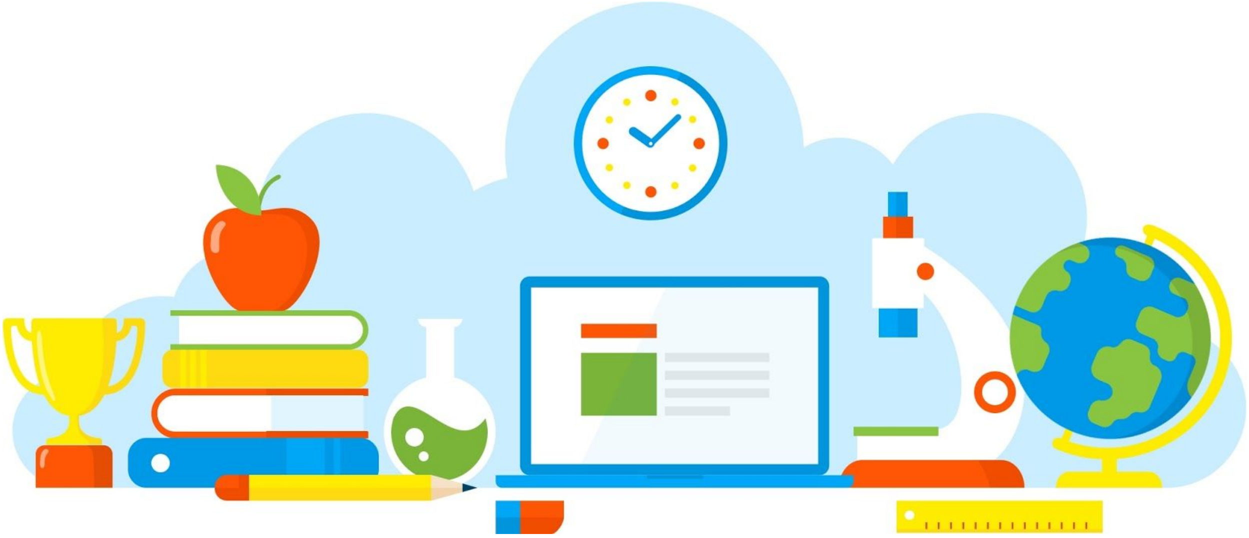

Mining Massive Datasets Homework 3

Teacher: Dr. Haratizadeh

Mahdi Sadeghi
830598035
1398/9/16

Question 1. (Flajolet Martin Algorithm)

To implement stream algorithms, I've developed an abstract base class which all the stream algorithms shall inherit. It has two abstract functions which should be implemented by it's children. One is "feed" that takes the stream data 1 by 1 and runs the algorithm on it. The other one is "get_result" that is used after the stream ends to retrieve the end result.

```
class StreamAlgorithm(ABC):
   @abstractmethod
   def feed(self, data):
       pass

   @abstractmethod
   def get_result(self, *args, **kwargs):
       pass

   def feed_dataset(self, dataset):
       for _, data in dataset.iterrows():
           self.feed(data)
```

The Flajolet Martin Algorithm is a rather simple approach to guess number of distinct items in the set, when saving all the different accord instances of data is not applicable. In this algorithm we simply hash each item and return $2^{max(number\ of\ zeros\ on\ tail\ of\ hash)}$ as result. We can also do multiple different hashes on the data and have different guesses. Then splitting the guesses in buckets and take average of them. At last the result would be the median of these averages. The code implementation is as follows:

```
class FlajoletMartinAlgorithm(StreamAlgorithm):
   def __init__(self, column, k):
       # column specifies which column to hash
       self.column = column
       # k is the number different hashes to use
       self.k = k
       # max_zeros is used to save maximum number of zeros found on tail of
specific hash salt
```

```
        self.max_zeros = [0] * k

    def _hash_zero_count(self, value, salt=0):
        # prefix salt to the value to generate different hashes for different
salts
        # sha1 does not support direct salting
        mvalue = str(salt) + str(value)

        # hash with sha1
        a = hashlib.sha1(mvalue.encode()).hexdigest()

        # find and return number of zeros on the tail of hash
        zeros = 0
        for d in a[len(a)::-1]:
            bitarray = bin(int(d, 16))[2:]

            for bit in bitarray[len(bitarray)::-1]:
                if bit == '1':
                    break

                zeros += 1

            if d != '0':
                break

        return zeros

    def feed(self, data):
        # in this function we hash input data column with k different hash
functions and
        # save the maximum zeros found in the max_zeros array
        value = data[self.column]
        for i in range(self.k):
            zeros = self._hash_zero_count(value, i)
            self.max_zeros[i] = max(self.max_zeros[i], zeros)

    def get_result(self, f=5):
        # splits max_zeros array to "f" buckets and takes average of them
        bin_averages = [0] * f
        bin_sizes = [0] * f
        for i in range(self.k):
            bin_index = i % f
            bin_sizes[bin_index] += 1
            bin_averages[bin_index] = (bin_averages[bin_index] *
(bin_sizes[bin_index] - 1) + (
                    2 ** self.max_zeros[i])) / bin_sizes[bin_index]

        # sorts the averages and return the median
        sorted(bin_averages)
        return bin_averages[int(f / 2)]
```

Question 2. (Alon Matias Szegedy Algorithm)

In this algorithm, for each incoming element we track it by a probability. The number of tracked items cannot exceed a certain amount (s) for each stream. We want the 2nd moment for each region so we should convert the single input stream to multiple streams, one for each region. This allows us to compute moment for all of them separately. The implementation is as follows (the code is documented with comments):

```
class AlonMatiasSzegedyAlgorithm(StreamAlgorithm):
   def __init__(self, column, s, stream_detector):
       self.column = column
       # s: maximum number of samples we are allowed to have
       self.s = s
       # stream_detector: a function to generate different streams from a single stream
       self.stream_detector = stream_detector
       # tracked: a dictionary containing an array for each stream (stream is the key) the array \
       # contains tracked (element, value) tuples in which algorithm is working with
       self.tracked = {}
       # n: is a dictionary containing number of arrived items for each stream separately
       self.n = {}

   def _add_tracked_item(self, stream, data):
       target_element = data[self.column]
       self.tracked[stream].append([target_element, 0])

   def _remove_tracked_item(self, stream, index):
       del self.tracked[stream][index]

   def _update_tracked_items(self, stream, data):
       # this function gets a data and increments value of all the tracked items with same element
       target_element = data[self.column]
       for i, tracked_item in enumerate(self.tracked[stream]):
           element, value = tracked_item
           if element == target_element:
               self.tracked[stream][i][1] += 1

   def _get_tracked_items(self, stream):
       if stream not in self.tracked:
           self.tracked[stream] = []
           self.n[stream] = 0

       return self.tracked[stream]

   def feed(self, data):
       # detect the stream
       stream = self.stream_detector(data)

       # check if you should pick this specific data
       tracked_items = self._get_tracked_items(stream)
       # if we haven't filled s positions for this stream, pick it
       if len(tracked_items) < self.s:
           self._add_tracked_item(stream, data)
       else:
           # if we are at max (s items tracked) select it with probability of
s/n+1
           selection_coin = random.randrange(0, self.n[stream] + 1)
           if selection_coin < self.s:
               # now remove one of the previous items with probability of 1/s
               removed_item = random.randrange(0, self.s)
               self._remove_tracked_item(stream, removed_item)
               self._add_tracked_item(stream, data)

       self.n[stream] += 1
       self._update_tracked_items(stream, data)

   def get_result(self, m):
       # returns "m"th moment of all the streams
       results = {}
       # compute v^m - (v-1)^m for all streams
       for stream in self.tracked:
           results[stream] = 0
           for element, value in self.tracked[stream]:
               results[stream] += (value ** m) - ((value - 1) ** m)

       return results
```

For using this class we must specify a stream_detector function that gets an element and returns a string indicating it’s stream which in this question is a function that simply returns the elements region:

```
ams1 = AlonMatiasSzegedyAlgorithm('goods', 10, lambda x: x['region'])
ams1.feed_dataset(dataset)
momend_2_per_region = ams1.get_result(2)
print('AlonMatiasSzegedy Algorithm Result (2nd Moment Per Region):
{0}'.format(momend_2_per_region))
```

Question 3. (Alon Matias Szegedy Algorithm)

The only difference between solution to question 2 and 3 is the stream detector which would be a combination of region and month for question 3:

```
ams2 = AlonMatiasSzegedyAlgorithm('goods', 10, lambda x: x['region'] + ':' +
x['date'].split(' ')[1])
ams2.feed_dataset(dataset)
momend_2_per_region_month = ams2.get_result(2)
print('AlonMatiasSzegedy Algorithm Result (2nd Moment Per Region and Month):
{0}'.format(momend_2_per_region_month))
```