MapReduce

Μέρος 1ο: Εξαγωγή Πληροφορίας - SQL

1Α. Εξαγωγή πληροφορίας με διαφορετικούς τρόπους

1Α.1. Q1 με χρήση MapReduce κώδικα

```
map(key, value):
    #key: None
    #value: csv line
    line = value.split(",")
    start_time = strptime(line[1], "%Y-%m-%d %H:%M:%S"))
    hour = start_time.hour
    start_hour = extract_hour(start_datetime)
    longitude = line[3]
    latitude = line[4]
    emit(hour, (longitude, latitude, 1))

reduce(key, value):
    #key: hour
    #value: a list of (longitude, latitude, 1) tuples
    hour = key
    sum_longitude = 0
    sum_latitude = 0
    count = 0
    for v in value:
        sum_longitude = sum_longitude + value[0]
        sum_latitude = sum_latitude + value[1]
        count = count + value[2]
    emit(hour, (sum_longitude, sum_latitude, count))

map(key, value):
    #key: None
    #value: a tuple of (hour, (sum_longitude, sum_latitude, count))
    hour = value[0]
    sum_longitude = value[1][0]
    sum_latitude = value[1][1]
    count = value[1][2]
    avg_longitude = sum_longitude / count
    avg_latitude = sum_latitude / count
    emit(hour, avg_longitude, avg_latitude)
```

Listing 1: Q1 MapReduce

1Α.2. Q2 με χρήση MapReduce κώδικα

```
#yellow_tripvendors_1m RDD
map(key, value):
    #key: None
    #value: csv line
    line = value.split(",")
    id = line[0]
    vendor = line[1]
    emit(id, vendor)

#yellow_tripdata_1m RDD
map(key, value):
    #key: None
    #value: csv line
    line = value.split(",")
    id = line[0]
    datetime_start = strptime(line[1], "%Y-%m-%d %H:%M:%S")
    datetime_end = strptime(line[2], "%Y-%m-%d %H:%M:%S")
    duration = (datetime_end - datetime_start).total_seconds() / 60.0
    start_longitude = line[3]
    start_latitude = line[4]
    end_longitude = line[5]
    end_latitude = line[6]
    distance = haversine(start_longitude, start_latitude, end_longitude, end_latitude)
    emit(id, [distance, duration])

#joined RDDs
map(key, value):
    #key: None
    #value: a tuple of (id, ([distance, duration], vendor))
    duration = value[1][0][0]
    distance = value[1][0][1]
    vendor = value[1][1]
    emit(vendor, duration, distance)

map(key, value):
    #key: None
    #value: a tuple of (vendor, distance, duration)
    vendor = value[0]
    emit(vendor, value)

reduce(key, value):
    #key: vendor
    #value: a list of (vendor, distance, duration) tuples
    vendor = key
    max_distance = value[0][1]
    duration = value[0][2]
    for v in value:
        if v[1] > max_distance:
            max_distance = v[1]
            duration = v[2]
    emit(vendor, (max_distance, duration))
```

Μέρος 2ο: Machine Learning - Κατηγοριοποίηση κειμένων

Πρώτος Τρόπος Επίλυσης

```
#customer_complaints RDD
map(key, value):
    #key: None
    #value: csv line
    line = value.split(",")
    date = line[0]
    productCategory = line[1]
    complaints = line[2].lower()
    emit(productCategory, complaints)

flatmap(key, value):
    #key: None
    #value: a tuple of (productCategory, complaints)
    words = value[1].split(" ")
    emit(words)

map(key, value):
    #key: None
    #value: word
    word = re.sub('[^a-zA-Z]+', '', value)
    emit(word)

map(key, value):
    #key: None
    #value: word
    emit(value, 1)

reduce(key, value):
    #key: word
    #value: a list of 1's
    word = key
    counter = 0
    for v in value:
        counter = counter + v
    emit(word, counter)

map(key, value):
    #key: None
    #value: a tuple of (word, counter)
    word = value[0]
    emit(word)

#customer_complaints RDD
map(key, value):
    #key: None
    #value: a tuple of (productCategory, complaints)
    string_label = value[0]
    words_list = complaints.split(" ")
    emit(string_label, words_list)
```

```
map(key, value):
    #key: None
    #value: a tuple of (string_label, words_list)
    string_label = value[0]
    words_list = value[1]
    list_of_sentence_words_in_lexicon = [word for word in words_list if
                                         word in broad_com_words]
    emit(string_label, list_of_sentence_words_in_lexicon)

#idf RDD
flatmap(key, value):
    #key: None
    #value: ((string_label, list_of_sentence_words_in_lexicon), sentence_index)
    words = set(value[0][1])
    for word in words:
        emit(word, 1)

reduce(key, value):
    #key: word
    #value: a list of 1's
    word = key
    counter = 0
    for v in value:
        counter = counter + v
    emit(word, counter)

map(key, value):
    #key: None
    #value: a tuple of (word, counter)
    word = value[0]
    counter = value[1]
    idf = log(number_of_complaints/counter)
    emit(word, idf)

#customer_complaints RDD
flatmap(key, value):
    #key: None
    #value: a tuple of ((string_label, list_of_sentence_words_in_lexicon),
    sentence_index)
    string_label = value[0][0]
    list_of_sentence_words_in_lexicon = value[0][1]
    sentence_index = value[1]
    for word in list_of_sentence_words_in_lexicon:
        emit((word, string_label, sentence_index), 1)

reduce(key, value):
    #key: (word, string_label, sentence_index)
    #value: a list of 1's
    word_count_in_sentence = 0
    for v in value:
        word_count_in_sentence = word_count_in_sentence + v
    emit(key, word_count_in_sentence)

map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index, sentence_length),
    word_count_in_sentence)
```

```
    word = value[0][0]
    string_label = value[0][1]
    sentence_index = value[0][2]
    word_count_in_sentence = value[1]
    word_index_in_lexicon = broad_com_words.value.index(word)
    emit(word, (string_label, sentence_index, word_count_in_sentence,
    word_index_in_lexicon))

#joined
map(key, value):
    #key: None
    #value: a tuple of (word, ((string_label, sentence_index,
                        word_count_in_sentence, word_index_in_lexicon),
                        idf))
    word = value[0]
    string_label = value[1][0][0]
    sentence_index = value[1][0][1]
    word_count_in_sentence = value[1][0][2]
    idf = value[1][1]
    word_index_in_lexicon = value[1][0][3]
    emit((word, string_label, sentence_index),
         (word_count_in_sentence, idf, word_index_in_lexicon))

#tf RDD
map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index),
                        (word_count_in_sentence, idf, word_index_in_lexicon))
    string_label = value[0][1]
    sentence_index = value[0][2]
    word = value[0][0]
    word_count_in_sentence = value[1][0]
    emit((string_label, sentence_index), ([word], [word_count_in_sentence]))

reduce(key, value):
    #key: (string_label, sentence_index)
    #value: a list of ([word], [word_count_in_sentence]) tuples
    words_list = [] #listof(words)
    word_count_list = [] #listof(word_count_in_sentence)
    for v in value:
        word = v[0]
        word_count_in_sentence = v[1]
        words_list.append(word)
        word_count_list.append(word_count_in_sentence)
    emit(key, (words_list, word_count_list))

map(key, value):
    #key: None
    #value: a tuple of ((string_label, sentence_index),
                        (words_list, word_count_list))
    string_label = value[0][0]
    sentence_index = value[0][1]
    words_list = value[1][0] #listof(words)
    word_count_list = value[1][1] #listof(word_count_in_sentence)
    result_list = [] #listof(word_count/max_word_count)
    for word_count in word_count_list:
        result_list.append(word_count/get_max(word_count_list))
```

```
    emit((string_label, sentence_index), (words_list, result_list))

map(key, value):
    #key: None
    #value: a tuple of ((string_label, sentence_index), (words_list, result_list))
    (x[0], (x[1][0], [(0.5 + 0.5 * y) for y in x[1][1]]))
    string_label = value[0][0]
    sentence_index = value[0][1]
    words_list = value[1][0] #listof(words)
    result_list = value[1][1] #listof(word_count/max_word_count)
    tf_list = [] #listof(tf)
    for y in result_list:
        tf_list.append(0.5 + 0.5 * y)
    emit((string_label, sentence_index), (words_list, tf_list))

map(key, value):
    #key: None
    #value: a tuple of ((string_label,sentence_index), (words_list, tf_list))
    string_label = value[0][0]
    sentence_index = value[0][0]
    words_list = value[1][0] #listof(words)
    tf_list = value[1][1] #listof(tf)
    emit((string_label, sentence_index), [words_list, tf_list])

flatmap(key, value):
    #key: None
    #value: a tuple of ((string_label, sentence_index), [words_list, tf_list])
    words_list = value[1][0] #listof(words)
    for i in range(0, length(words_list))
        word = value[1][0][i]
        string_label = value[0][0]
        sentence_index = value[0][1]
        tf = value[1][1][i]
        emit((word, string_label, sentence_index), tf)

#joined
map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index),
                        ((word_count_in_sentence, idf, word_index_in_lexicon),
                        tf))
    word = value[0][0]
    string_label = value[0][1]
    sentence_index = value[0][2]
    word_index_in_lexicon = value[1][0][2]
    idf = value[1][0][1]
    tf = value[1][1]
    tfidf = idf * tf
    emit((word, string_label, sentence_index), (word_index_in_lexicon, tfidf)

map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index),
                        (word_index_in_lexicon, tfidf)
    sentence_index = value[0][2]
    string_label = value[0][1]
    word_index_in_lexicon = value[1][0]
```

```
    tfidf = value[1][1]
    emit((sentence_index, string_label), [(word_index_in_lexicon, tfidf)])

reduce(key, value):
    #key: (sentence_index, string_label)
    #value: a list of (word_index_in_lexicon, tfidf) tuples
    list_of_values = [] #listof((word_index_in_lexicon, tfidf))
    for v in value:
        list_of_values.append(v)
    emit(key, list_of_values)

map(key, value):
    #key: None
    #value: a tuple of ((sentence_index, string_label), list_of_values)
    string_label = value[0][1]
    list_of_values = value[1] #listof((word_index_in_lexicon, tfidf))
    sorted_on_word_index_in_lexicon_list = sort(list_of_values, by key = value[1][0])
    emit(string_label, sorted_on_word_index_in_lexicon_list)

map(key, value):
    #key: None
    #value: a tuple of (string_label, sorted_on_word_index_in_lexicon_list)
    string_label = value[0]
    sorted_on_word_index_in_lexicon_list = value[1] #listof((word_index_in_lexicon,
                                                            tfidf))
    word_index_list = [] #list_of(word_index_in_lexicon)
    tfidf_list = [] #list_of(tfidf)
    for word_index_in_lexicon, tfidf in
        sorted_on_word_index_in_lexicon_list:
            word_index_list.append(word_index_in_lexicon)
            tfidf_list.append(tfidf)
    emit(string_label, SparseVector(lexicon_size, word_index_list, tfidf_list))
```

Δεύτερος Τρόπος Επίλυσης

```
#customer_complaints RDD
flatmap(key, value):
    #key: None
    #value: a tuple of (productCategory, complaints)
    words = value[1].split(" ")
    emit(words)

map(key, value):
    #key: None
    #value: word
    word = re.sub('[^a-zA-Z]+', '', value)
    emit(word)

map(key, value):
    #key: None
    #value: word
    emit(value, 1)

reduce(key, value):
    #key: word
    #value: a list of 1's
    word = key
    counter = 0
    for v in value:
        counter = counter + v
    emit(word, counter)

map(key, value):
    #key: None
    #value: a tuple of (word, counter)
    word = value[0]
    emit(word)

map(key, value):
    #key: None
    #value: a tuple of (productCategory, complaints)
    string_label = value[0]
    words_list = complaints.split(" ")
    emit(string_label, words_list)

map(key, value):
    #key: None
    #value: a tuple of (string_label, words_list)
    string_label = value[0]
    words_list = value[1]
    list_of_sentence_words_in_lexicon = [word for word in words_list if
                                         word in broad_com_words]
    emit(string_label, list_of_sentence_words_in_lexicon)

#idf RDD
flatmap(key, value):
    #key: None
    #value: ((string_label, list_of_sentence_words_in_lexicon), sentence_index)
    words = set(value[0][1])
    for word in words:
```

```
        emit(word, 1)

reduce(key, value):
    #key: word
    #value: a list of 1's
    word = key
    counter = 0
    for v in value:
        counter = counter + v
    emit(word, counter)

map(key, value):
    #key: None
    #value: a tuple of (word, counter)
    word = value[0]
    counter = value[1]
    idf = log(number_of_complaints/counter)
    emit(word, idf)

#customer_complaints RDD
flatmap(key, value):
    #key: None
    #value: a tuple of ((string_label, list_of_sentence_words_in_lexicon),
    sentence_index)
    string_label = value[0][0]
    list_of_sentence_words_in_lexicon = value[0][1]
    sentence_length = length(list_of_sentence_words_in_lexicon)
    sentence_index = value[1]
    for word in list_of_sentence_words_in_lexicon:
        emit((word, string_label, sentence_index, sentence_length), 1)

reduce(key, value):
    #key: (word, string_label, sentence_index, sentence_length)
    #value: a list of 1's
    word_count_in_sentence = 0
    for v in value:
        word_count_in_sentence = word_count_in_sentence + v
    emit(key, word_count_in_sentence)

map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index, sentence_length),
    word_count_in_sentence)
    word = value[0][0]
    string_label = value[0][1]
    sentence_index = value[0][2]
    word_count_in_sentence = value[1]
    sentence_length = value[0][3]
    word_frequency_in_sentence = word_count_in_sentence/sentence_length
    word_index_in_lexicon = broad_com_words.value.index(word)
    emit(word, (string_label, sentence_index, word_frequency_in_sentence,
    word_index_in_lexicon))

#joined
map(key, value):
    #key: None
    #value: a tuple of (word, ((string_label, sentence_index,
```

```
                        word_frequency_in_sentence, word_index_in_lexicon),
                        idf))
    word = value[0]
    string_label = value[1][0][0]
    sentence_index = value[1][0][1]
    word_frequency_in_sentence = value[1][0][2]
    idf = value[1][1]
    tfidf = word_frequency_in_sentence*idf
    word_index_in_lexicon = value[1][0][3]
    emit((word, string_label, sentence_index), (tfidf, word_index_in_lexicon))

map(key, value):
    #key: None
    #value: a tuple of ((word, string_label, sentence_index),
                        (tfidf, word_index_in_lexicon))
    sentence_index = value[0][2]
    string_label = value[0][1]
    word_index_in_lexicon = value[1][1]
    tfidf_in_sentence = value[1][0]
    emit((sentence_index, string_label), [(word_index_in_lexicon, tfidf_in_sentence)])

reduce(key, value):
    #key: None
    #key: (sentence_index, string_label)
    #value: a list of (word_index_in_lexicon, tfidf_in_sentence) tuples
    list_of_values = [] #listof((word_index_in_lexicon, tfidf_in_sentence))
    for v in value:
        list_of_values.append(v)
    emit(key, list_of_values)

map(key, value):
    #key: None
    #value: a tuple of ((sentence_index, string_label), list_of_values)
    string_label = value[0][1]
    list_of_values = value[1] #listof((word_index_in_lexicon, tfidf_in_sentence))
    sorted_on_word_index_in_lexicon_list = sort(list_of_values, by key = value[1][0])
    emit(string_label, sorted_on_word_index_in_lexicon_list)

map(key, value):
    #key: None
    #value: a tuple of (string_label, sorted_on_word_index_in_lexicon_list)
    string_label = value[0]
    sorted_on_word_index_in_lexicon_list = value[1] #listof((word_index_in_lexicon,
                                                            tfidf_in_sentence))
    word_index_list = [] #list_of(word_index_in_lexicon)
    tfidf_list = [] #list_of(tfidf_in_sentence)
    for word_index_in_lexicon, tfidf_in_sentence in
        sorted_on_word_index_in_lexicon_list:
            word_index_list.append(word_index_in_lexicon)
            tfidf_list.append(tfidf_in_sentence)
    emit(string_label, SparseVector(lexicon_size, word_index_list, tfidf_list))
```