Medical Bioinformatics

Project

SS15

Stefan Jacob & Zarin Shakibaei

Outline

1. Introduction
2. Data
3. Methods
4. Results
5. Conclusion
6. Introduction
7. Data

Two different data archives have been downloaded.

The first archive comes from the Genome Characterization Center (GCC) and contains txt files with gene samples and their corresponding expression level (log2 lowess normalized). The second archive which is provided by Biospecimen Core Resource(BCR) contains carefully cataloged tissue and sample information with important medical information about the patient. Each text file from the BCR contains other information e.g. the gender of the patient or the treatment, used drug and some information about the tumor (size, site, location, etc.)

Among the biotab txt files, eight contained useful information which have the following names:

nationwidechildrens.org\_biospecimen\_cqcf\_luad.txt

*(contains information about the consent or death status of the patient we could use this as a sign how lethal the tumor is?)*

nationwidechildrens.org\_biospecimen\_tumor\_sample\_luad.txt

*(contains information about the tumor sample weight and tumor necrosis percentage(?))*

nationwidechildrens.org\_clinical\_cqcf\_luad.txt

*(contains information about tumor sample country and site of primary tumor)*

nationwidechildrens.org\_clinical\_follow\_up\_v1.0\_luad.txt

*(contains information about the vital status and the tumor status of the patient, the treatment outcome of the first course, information if a new tumor event occurred and the followup treatment success)*

nationwidechildrens.org\_clinical\_omf\_v4.0\_luad.txt

*(contains information about other malignancies, their anatomic site and the histological type)*

nationwidechildrens.org\_clinical\_patient\_luad.txt

*(contains information about the patient’s gender, if the patient was a smoker and how much he smokes, the pathologic stage of the tumor)*

nationwidechildrens.org\_clinical\_radiation\_luad.txt

*(contains information about the used radiation method and their duration, etc.)*

As mentioned previously, the downloaded GCC-Data folder, contains 35 txt files, each referring to one patient. The first column contains all Genes from Microarry chip, the second column are expression levels corresponding to each gene.

1. Methods

3.0. Idea

After mature consideration, we decided to take above-mentioned transcriptomic data from TCGA’s Genome Characterization Center (GCC) and joined it in a big table with Flinks TableAPI (see below). The problem now was that we had a table with 17,816 lines — one for each gene and its expression level. We came up with the idea to preprocess the data in that way, that we only wanted to keep genes that showed a significant change in their expression value over the different patients. Therefore, we tried to use the standard deviation for each line of the table and only take lines into account, that had a sd bigger than for example 3. The occurring problem was that Flinks TableAPI was only capable of taking the sd of columns into account, but not the one of lines. That’s why we transposed the dataset with the expression levels resulting in a set that had 17,816 columns. Now the problem was that we had to enumerate these 17,816 columns in order to be able to read in the dataset with Flinks TableAPI as each unique column had to the mentioned (we found no way to simply read in all columns without telling Flink how many columns there exists…). So that’s where we are stuck at the moment…

* 1. Read in all data

The goal here was to read all files in the GCC folder and join them so that at the end we get one single file, which contains one Gene column and 35 Gene Expression Level of all 35 patients.

*// function to list all files in a directory***def** getListOfFiles(dir: String): List[File] = {  
 **val** d = **new** File(dir)  
 **if** (d.exists && d.isDirectory) {  
 d.listFiles.filter(\_.isFile).toList  
 } **else** {  
 *List*[File]()  
 }  
}

*// extract the filename of absolute path to file***val** files = getListOfFiles(*dataGCCFilePath*)  
**val** filenameArray = files.toString.split(**","**)  
**val** sizeOfFilenameArray = filenameArray.size

*//val items = getDataSetFile(env,filenameArray(1)).as('firstFileCol1, 'firstFileCol2)***val** firstFile = *getDataSetFile*(env,filenameArray(1)).as(**'firstFileCol1**, **'firstFileCol2**)  
**for** (i <- 2 to filenameArray.size-2) {  
 *// Read a file but only includes the 1st, 2nd column - returns DataSet[MyLineitem]* **val** CurrentFile = *getDataSetFile*(env,filenameArray(i)).as(**'col1**, **'col2**)  
  
 **val** items =  
 firstFile.join(CurrentFile)  
 .where(**'firstFileCol1** === **'col1**)  
 .select()  
  
}  
items.writeAsCsv(**"file://path"**, **"\n"**, **"\t"**)

This method has been failed due to a .writeAsCsv() method problem.

Somehow this method doesn’t recognize items variable out of the fore

Loop. In addition the way joining works in Flink, would never let us to join tables in this way. Flink can always join two fixed tables but one of our tables was growing over the time.

* 1. Read in Data new approach

After reconsidering all available approaches, it remained

only one way to handle our input data, using Interface InputFormat<OT,T extends [InputSplit](https://ci.apache.org/projects/flink/flink-docs-master/api/java/org/apache/flink/core/io/InputSplit.html)> (text, file InputFormat) then define a [**createInputSplits**](https://ci.apache.org/projects/flink/flink-docs-master/api/java/org/apache/flink/api/common/io/InputFormat.html#createInputSplits(int))

so that every file is an input split, one record per gene symbol.

Write a Join Function

<https://ci.apache.org/projects/flink/flink-docs-master/apis/dataset_transformations.html#filter>

* 1. Machine Learning

*“perform a number of transformations to that data, and then output the transformed data, either to be used as the input (features) of a predictor function, such as a learning model, or just output the transformed data themselves, to be used in some other task.”*

[Supervised learning](https://en.wikipedia.org/wiki/Supervised_learning)*: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that*[maps](https://en.wikipedia.org/wiki/Map_(mathematics))*inputs to outputs.*

*Another categorization of machine learning tasks arises when one considers the desired output of a machine-learned system:*[[3]](https://en.wikipedia.org/wiki/Machine_learning#cite_note-bishop-3)*:3*

*In*[classification](https://en.wikipedia.org/wiki/Statistical_classification)*, inputs are divided into two or more classes, and the learner must produce a model that assigns unseen inputs to one (or*[multi-label classification](https://en.wikipedia.org/wiki/Multi-label_classification)*) or more of these classes. This is typically tackled in a supervised way. Spam filtering is an example of classification, where the inputs are email (or other) messages and the classes are "spam" and "not spam".*

*In*[machine learning](https://en.wikipedia.org/wiki/Machine_learning)*,****support vector machines****(****SVMs****, also****support vector networks***[[1]](https://en.wikipedia.org/wiki/Support_vector_machine#cite_note-CorinnaCortes-1)*) are*[supervised learning](https://en.wikipedia.org/wiki/Supervised_learning)*models with associated learning*[algorithms](https://en.wikipedia.org/wiki/Algorithm)*that analyze data and recognize patterns, used for*[classification](https://en.wikipedia.org/wiki/Statistical_classification)*and*[regression analysis](https://en.wikipedia.org/wiki/Regression_analysis)*. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-*[probabilistic](https://en.wikipedia.org/wiki/Probabilistic_classification)[binary](https://en.wikipedia.org/wiki/Binary_classifier)[linear classifier](https://en.wikipedia.org/wiki/Linear_classifier)*.*

Regarding