BIOINFORMATICS

Vol. 00 no. 00 2015
Pages 1–5

A workflow for the identification of breast cancer inducing genes

Niklas Wulkow,[1] *

[1]Freie Universität Berlin

ABSTRACT

Motivation: With millions of genes and proteins, the human body provides tons of data that can be analyzed for various means. Detecting and identifying disease-causing genes and proteins is one of them. Multiple different approaches have been made therefore, from regression to network analysis. In this paper I will present some of them. I have also included them into an analysis 'pipeline', which can be fed with data and produces results as to which genes are candidates to be disease-causing. To be precise, the disease we are looking at is breast cancer ('Breast invasive carcinoma,' BRCA).

Results: I will name genes that are, according to my pipeline, candidates to be disease-causing at the end of this article and evaluate the results.

Availability and implementation: My code is available at https://github.com/nwulkow/FlinkProject.
It is written in Scala.

Contact: niklas.wulkow@ewetel.net

1 INTRODUCTION:

My analysis pipeline for gene data uses various methods in order to find genes that cause breast cancer. I will explain one machine learning and two network-based approaches and how I included them into the pipeline.
This article is structured the following way: I will give an overview about the data used and the methods I built in, describe the pipeline in a more detailed way and talk about the output of the pipeline in the end.

2 DATA:

The data that was used was taken from 'The Cancer Genome Atlas' (TCGA). This database offers real data contained in .txt-files for free. I ran my pipeline on two types of data: miRNA-data and mRNA-data. The miRNA-data consists of several datasets with 1046 features each, i.e. 1046 miRNA-codes and their corresponding frequency in the blood probe taken from a person. The mRNA-data contains 20502 features per person.

As every data package of mRNA and miRNA contains more than one type of data, I ignored some files and only considered the remaining ones. For miRNA, the used files have the ending '.mirna.quantification' and for mRNA, the used files end on 'rsem.genes.results'.

*to whom correspondence should be addressed

They are structured the following way: For miRNA, every .txt-file has four columns, the `miRNA-ID`, the `read_count`, `reads_per_million_miRNA_mapped` and `cross_mapped`. The first and third columns are the ones we are interested in. For mRNA, the .txt-files have the four columns: `gene_id`, `raw_count`, `scaled_estimate` and `transcript_id`. The lines 2-30 of the mRNA-files also have to be erased. Fortunately, the entire reading process is done in the pipeline so the user does not have to change anything inside the files.

2.1 BIG data

An important attribute of the data is its size. Handling multiple data points with thousand of entries each can provide problems. Running a not well constructed algorithm on the data on a slow computer can take far too long for the user to accept. Due to that, it needs to be made sure that the workflow is constructed in a smart and efficient way.
On top of that, there is Flink, an 'open source platform for scalable batch and stream data processing' (from https://flink.apache.org). It provides methods and a network that enables users and developers to work on very large datasets very quickly. Its developers describe it as 'fast, easy to use, reliable, scalable, expressive' and 'hadoop-compatible'. It provides useful 'API's' (application programming interfaces) for machine learning and graph analysis that I both incorporated into my pipeline.
It also provides a network cluster which a program can be run on. The program is then supposed to be executed much faster compared to it being run it on a local machine.
Further, the Flink-algorithms are written in Scala, a programming language whose advantage over other programming languages lies in the fact that it can deal with big datasets quicker. Hence, the pipeline was written in Scala.

3 METHODS:

I have included four Flink-based algorithms into my pipeline: Matrix Completion, SVM (Support Vector Machines), PageRank and Community Detection.

3.1 Matrix Completion:

Matrix completion is a topic that has come up only a few years ago. It addresses the attempt to figure out all entries of a matrix even if only a certain fraction of them is given. Different algorithms have been developed that tackle this problem and one of them

has already been implemented in Scala. Here is a description of the algorithm, taken from https://ci.apache.org/projects/flink/flink-docs-master/libs/ml/als.html:

'The alternating least squares (ALS) algorithm factorizes a given matrix R into two factors U and V such that $R \approx UV$. The unknown row dimension is given as a parameter to the algorithm and is called latent factors. Since matrix factorization can be used in the context of recommendation, the matrices U and V can be called user and item matrix, respectively. The i-th column of the user matrix is denoted by$u_i$and the i-th column of the item matrix is v_i. The matrix R can be called the ratings matrix with $R_{ij} = r_{ij}$.
In order to find the user and item matrix, the following problem is solved:

$argmin_{U,V} \sum_{i,j|r_{ij} \neq 0} (r_{ij} u_i v_j)^2 + \lambda(\sum_i \|u_i\|^2 + \sum_j \|v_j\|^2)$

with λ being the regularization factor, u_i being the number of items the user i has rated and v_j being the number of times the item j has been rated. This regularization scheme to avoid overfitting is called weighted-λ-regularization. Details can be found in the work of Zhou et al..

By fixing one of the matrices U or V, we obtain a quadratic form which can be solved directly. The solution of the modified problem is guaranteed to monotonically decrease the overall cost function. By applying this step alternately to the matrices U and V, we can iteratively improve the matrix factorization.'

Matrix completion requires the matrix to have a low-rank-structure. The matrix that contains the data (the i,j-th entry of the matrix represents the frequency of the j-th gene in the i-th blood probe) is expected to have low rank or to be close to it, since the concentration of a certain gene in a blood probe should be similar among all healthy respectively all diseased people.

Matrix completion can be useful for our pipeline, since real data is far from perfect. Chances are that it is not complete, so that we have to make it complete by guessing what the missing entries are. For that aim, matrix completion is the perfect tool.

3.2 SVM

SVM (Support Vector machines) is a mathematical method that can be used to seperate datapoints into two sets by a hyperplane. SVM describes an optimization problem whose solution classifies certain points into one group and the remaining points into the other group. The solution itself is a vector whose dimension equals the dimension of a datapoint. The optimization problems are made in a way such that an entry of that result vector, the 'classifier', has a high value if and only if there are big differences among classes regarding that particular entry.

SVM is useful for the detection of disease-causing genes, because by comparing the gene data of healthy and diseased people and applying SVM, the resulting classifier indicates which genes are important. Using that classifier, we can thaen make a statement about whether a person whose health we are not informed about is diseased or not.

A typical SVM optimazation problem looks like this:

minimize over all $\omega \in R^n$:

$\frac{1}{2}\|\omega\|_2^2 + C \sum_{i=1}^n max(0, 1 - y_i(\omega^T x_i + b))^2$

where x_i is the i-th row of the data matrix and y_i the 'label' which is 1 if the i-th person is healthy and -1 if not.

ω has the property: $w \cdot x_i > 0$ if x_i is the data vector of a healthy person and $w \cdot x_i < 0$ if not.

3.3 Pagerank

The PageRank algorithm orders all nodes of a graph by their 'importance' and gives them an according value.
On https://ci.apache.org/projects/flink/flink-docs-release-0.8.1/examples.html the following can be found about the PageRank algorithm:

'The PageRank algorithm computes the importance of pages in a graph defined by links, which point from one pages to another page. It is an iterative graph algorithm, which means that it repeatedly applies the same computation. In each iteration, each page distributes its current rank over all its neighbors, and compute its new rank as a taxed sum of the ranks it received from its neighbors. The PageRank algorithm was popularized by the Google search engine which uses the importance of webpages to rank the results of search queries.'

The 'pages' in our case are genes. So by applying the PageRank algorithm to our data to both subgroups, healthy and diseased, seperately, we get information on which are the most 'important' genes in one group.
The network we run PageRank on is the following: Each gene is represented by a node and there is an (unweighted and undirected) edge between two genes k and l if the absolute value of the correlation coefficient of the k-th and l-th column of the data matrix ($p(k, l)$) is higher than a certain threshold. This makes sense because a high correlation coefficient hints that a frequent occurance of one gene in the genom of a body leads to a high occurance of the other or vice versa.

3.4 Community Detection

The Community Detection algorithm is, as is the PageRank algorithm, part of the GellyAPI provided by Flink. It finds clusters, i.e. isolated or almost isolated components inside a network / graph. Applying it also both to the network built from the data from healthy people and diseased people we learn about which genes are 'linked' to each other. That could mean, if one gene has e.g. a lower occurance than usual than the same is likely to hold for exactly the genes that are in the same cluster.

4 THE PIPELINE

4.1 User arguments

The user can specify all parameters that occur in the pipeline beforehand. They can have influence on the speed and the results of the workflow. Those parameters are:

- Data
- Data that has to be classified by the SVM classifier
- Decide whether matrix completion is done or not
- The number of factors that are used in the matrix completion algorithm

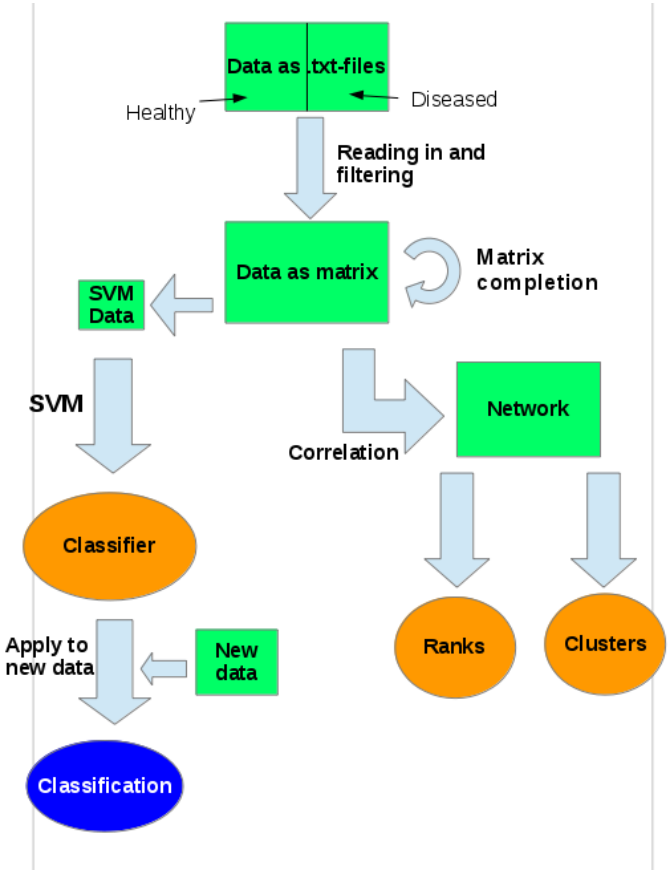

Fig. 1. Sketch of the pipeline.

- Number of iterations of Matrix Completion algorithm
- Number of iterations of SVM algorithm
- SVM regularization constant
- SVM algorithm stepsize
- Maximal number of Genes that is considered. If the user selects k as the maximal number of genes and the data contains more than k genes then only the first k genes from every blood probe are considered
- The threshold for the network construction. An edge is put between two nodes k and l if the correlation coefficient $p(k, l)$ is higher than that threshold
- Names of genes that are not to be considered

4.2 Input

The pipeline needs the path to the directory where the data is stored. The data of each group must be stored in a different folder. More information on that in the user description file.

4.3 Output

The output consists of 15 .txt-files. Seven of those contain information that the user of the pipeline might be interested in, i.e.:

- The classifier (genes with corresponding weights; ordered)
- Result of the classification of unknown data
- The PageRank and Community Detection results for both the healthy and diseased group stored in a .gdf-file which can be read by the software ‘Gephi‘
- A ‘rank differences‘ file I will give information about later on in this article

The remaining eight documents are temporary files that are used during the workflow: Sometimes data has to be parsed into a different format. To that end, it is written into .txt-files in the according style and read back into the pipeline at a later point in time.
The data that is meant here is:

- The data matrix before matrix completion since the matrix completion algorithm needs input-files
- The output of the matrix completion algorithm
- The network and nodes of both the healthy and the diseased group
- The data matrix stored in a way such that the SVM method can use it

4.4 Reading The Data

The first true step of the workflow is to read the data that it is supposed to work with. Under certain conditions, that can be an easy process. But as we do not know about the quality of our data we have to put in more effort to do that. As we want to keep alive the chance to apply matrix completion on the data, the data is parsed into a matrix.
The problem is: We cannot be sure that every file that we read data from contains information on the same genes. So we cannot simply read in the data from a file and store it in a row of the matrix.
What is hence done in the pipeline is the following: First of all a list of all genes that occur in the whole of the data is created. The list starts with all genes from the first file. If a new gene occurs in another file it is appended to the back end of the list.

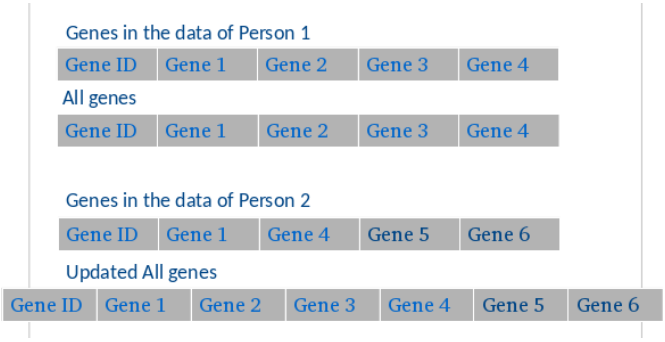

Fig. 2. Sketch of the creation of the data matrix. At first, a list of all genes that occur in the data is put together.

Afterwards we look through the data files for a second time: We consider a gene with corresponding value v, look for the index i of that gene ID in the gene list and store v in the i-th column (and according row) of the data matrix.

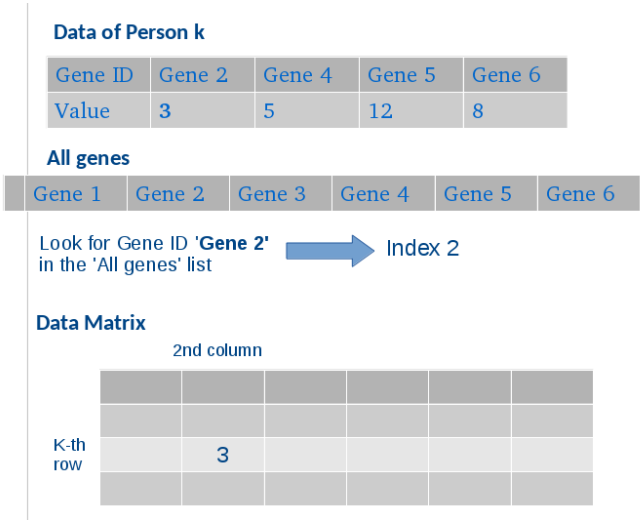

Fig. 3. Sketch of the creation of the data matrix. For a tuple of a gene and its corresponding value for particular person we look for the index of that gene in the genelist und store the value in the according column (and row) in the data matrix.

Also, the matrix is stored in a .txt-file in csv-format to be used by the matrix completion algorithm. As explained earlier, only files that have the correct ending (‘mirna.quantification‘ for miRNA and ‘rsem.genes.normalized_results‘ for mRNA) are used.
If the user has fixed a maximal number k of genes, we continue with only the first k columns of the matrix.

4.5 Matrix Completion

Matrix completion is optional. The user can decide for or against it in the programming arguments. If no matrix completion is done, missing values are represented with a 0 in the data matrix.

The output of the algorithm are .txt-files that contain the rows of the matrix factors U and V (see 3.1). Those factors have to be read from the output files. Eventually matrix multiplication of the two factors with each other needs to be done.
The resulting matrix then has the same format as the matrix that the matrix completion algorithm was fed with.

4.6 SVM

In order to apply an SVM method the data needs to be stored in a completely different format. The only way I found in which I can parse it into that format is the indirect way of writing it into another .txt-file in a certain way and then applying a common reading-method.
The SVM method needs each row of the data matrix as a vector with a corresponding ‘label‘. The label is 1 if the data vector belongs to a healthy person and -1 if it belongs to a diseased person.

4.6.1 Training After doing the SVM on the data, the result we get is the classifier that carries the weights of all genes. It partly depends on the tuning of the parameters.

4.6.2 Testing In the next step, we apply the classifier to data. We can classify people whose health is unknown to us (or simply test the accuracy of the classifier if we do know about the health) by looking at the scalar product of the classifier and the data vector (see 3.2).
The results of the classification and the classifier are then stored in a .txt-file.
For the read in of that data we proceed the same way as during the read in of the training data (Figure 4).

4.7 Network Analysis

As addressed earlier, the pipeline uses two different network analysis methods: PageRank and Community Detection.
Two networks are created: One for only the healthy people and one for only the diseased people. The networks and the results of the analysis methods on them can be compared later on.
During the network creation, for every pair of genes the correlationcoefficient p of the corresponding columns in the data matrix is computed. The mean and variance of every column of the data matrix (so the mean and variance of all values belonging to the same gene) are computed beforehand and plugged into the method that calculates the correlationcoefficient, so mean and variance only have to be computed once for each gene and not every time anew.
If the (absolute value of the) p-value is above a certain threshold, an (unweighted and undirected) edge is created between the two genes in the network. The network is stored in a .txt-file. Now the PageRank and Community Detection algorithms can be let loose on the data.
Their results are stored in .gdf-files, readable by Gephi. Additionally, for every gene the rank (not the PageRank value but actually how many genes there are that have a higher PageRank value) it was given from the PageRank algorithm for the healthy-people-network is compared to the one from the diseased-people-network: If a gene is in 300th place in the list of PageRanks for the healthy-people-network and in 800th in the other one, the distance is 500. The thought behind it is the following: If a gene plays a big role in the diseased-people-network (i.e. it has a high PageRank value there) and a small role in the healthy-people-network it is one to look at when it comes to the question which genes are disease causing. For all genes, this distance is written into the .txt-file with the name ’rank_differences’.

5 RESULTS

5.1 miRNA

I ran the pipeline on miRNA data of Breast Invasive Carcinoma, Batch 93 from the TCGA database. It contains 42 mirna.quantification-files from healthy and 30 from diseased people.
The five miRNA-IDs with the lowest (highest negative) value in the

classifier are: `hsa-mir-10b`, `hsa-mir-143`, `hsa-mir-10a`, `hsa-mir-22` and `hsa-mir-100`. The five miRNA-IDs with the highest positive value are: `hsa-mir-21`, `hsa-mir-182`, `hsa-mir-99b`, `hsa-mir-148a` and `hsa-mir-30a`.
The five miRNA-IDs with the highest rank difference are: `hsa-mir-198`,`hsa-mir-1933`,`hsa-mir-548c`,`hsa-mir-3676` and `hsa-mir-941-4`. However the top genes from the classifier do not have particularly high rank differences. It is questionable how meaningful the rank difference actually is.

As I ran the pipeline for a second time on the same data, the classifier was identical to the one from the the first run. That does not hold for the results of the network analysis though.

I used the very same data for testing of the classifier. Out of the 72 people, 56 were classified correctly. Applying the classifier to data from a different batch of the TCGA BRCA data, 69 out of 71 were classified correctly.

Running the workflow from start to finish took 1:18 minutes when executed locally.

I do not quite know how to interpret the miRNA results but this makes more sense with the mRNA results.

5.2 mRNA

Running the pipeline on mRNA data (20 healthy and 17 diseased people, batch 93 of the TCGA database), the top disease causing gene is - according to the classifier - AHNAK. The webpage www.proteinatlas.org gives information on how much a gene is associated with a certain disease by today's knowledge. It gives evidence that indeed AHNAK is well known to cause breast cancer. That however might be a lucky coincidence as it does not seem to be the case for all mRNA genes that have a high negative weight. It is not obvious that most of the mRNA genes with high negative weight are known to be diseasce-causing. The good news is that almost all those mRNA genes are not associated to different types of cancer if they are not associated to breast cancer, yet they tend to be if one actually is linked to breast cancer. On the other hand, according to proteinatlas.org and research papers that are published digitally, many of the mRNA genes with a high positive weight are also linked to breast cancer. All in all, the information the classifier provides does not match the information on proteinatlas.org in a satisfactory way.
Whereas the classifier managed to classify 10 out of 10 instances of test data (5 healthy, 5 diseased) from batch 96 correctly.

6 DISCUSSION AND OUTLOOK

As the discrepancies between the genes that the classifier states are important and results from the web suggest, universal correctness of results cannot be taken for granted. The fact that the classifier that was created in the pipeline provides relatively high accuracy emphasises that two equally correct statements can differ a lot.
The data itself surely is one major reason for that. Samples of the data, even data files from the TCGA database, might not mirror the reality exactly as e.g. the people who gave their blood probes may

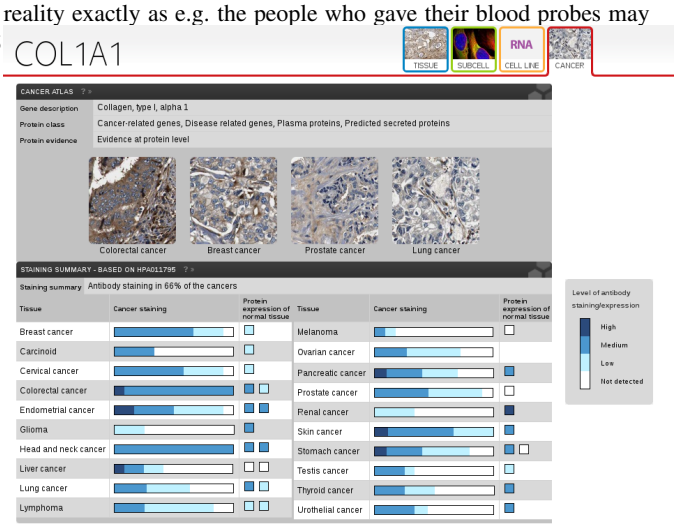

Fig. 4. proteinatlas.org gives insight on how much a gene is linked to different types of cancer.

have been much older than average. The experiments that create the results on pages like proteinatlas.org might not either. But it becomes obvious that the data that is worked with should be as close to the reality as possible.
Another reason for deviations between the results are parameters that are used in the methods. Even the starting point of an algorithm can influence the outcome of the method.
Fortunately, there are several completely different approaches to the task. In this article, three have been explained: The machine learning based SVM and the network analysis tools of PageRank and Community Detection. Surely there is great potential in them but as some of the methods are relatively new, experience has to be gained about how they behave under certain conditions, how to set the parameters and - most importantly - how to interpret the results.

In conclusion, the quest for disease-causing genes seems to be run at full throttle. Many different tools have helped research take big steps forward. As a human body carries tons of genes and with them tons of data, the handling of big data is an interesting point, too. The datasets quickly become too big to analysis them in short time on every computer. That problem is also being tackled, for example by Flink and their network cluster which a program can be run on very fast instead of running it on a local computer and taking much more time.
Cancer research with mathematical means is an interesting field that will surely develop further in the near future.