Machine Learning (CS-433) Project 2 Report

Camille Ohlmann, Michaël Spierer, Corentin Junod

Fall 2020

Abstract **– Flow cytometry is a tool widely used in biology to measure a number of parameters in a cell population. The cells are suspended in a fluid and their DNA is stained by a fluorescent dye. They are aligned in a flow by hydrodynamic focusing, then subsequently cross a laser beam, ideally one at a time. Their optical properties are then collected, and used to infer various characteristics. This paper uses Machine Learning tools applied to a flow cytometry data-set, measured on glacier samples by the Stream Biofilm and Ecosystem research Laboratory. It aims to separate cells from background and noise, so as to count them in each sample as accurately and reproducible as possible.**

1 Introduction

Climate change leads to the melting of glaciers and ice sheets, also affecting downstream aquatic ecosystem such as glacier-fed streams. At SBER (Stream Biofilm and Ecosystem research Laboratory), a team of scientists collects glacier-fed stream water and sediment samples from across the world to analyse their microbiomes before they are lost. To this end, this laboratory is interested in counting the number of cells in these samples by flow cytometry.

However, the low number of bacterial cells and the high amount of background noise affect counting efficiency as it involves differentiating cell events from noise events. This is typically done manually by the individual scientists, which can be somewhat tedious and prone to subjectivity and misclassification, and a Machine Learning based approach seems appropriate. As there are no labeled training data, we used Unsupervised Learning algorithms to build models that best address this issue. To exploit these models to their fullest and increase their accuracy, we also made several adjustments to the data and model parameters.

This paper aims at illustrating the steps and choices that were made to achieve our results. We will first explain how the cytometry data is collected and how these results are to be interpreted intuitively. Then we will discuss how we handled the different data files we were provided with, and explain how we processed the data before we used it. Then we will go into the details of the different algorithms and methods we used, while simultaneously giving the results obtained for each of them.

2 Cytometry data interpretation

As shown in Figure 1, flow cytometry works as follows: cell DNA is stained, cells are brought into suspension in a fluid, and then they are separated using flow techniques to go through a thin tube, ideally one by one. The dye binding to the DNA of each cell is then excited by the laser light and, using optical filters, several parameters related to scattering and fluorescence properties are measured for each cell. This is done very rapidely (several hundreds of cells per second).

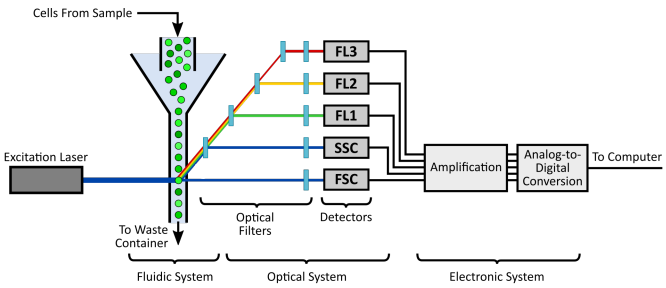


Figure 1: Flow Cytometry

This generates 5 different outputs, each divided in 2 features that represent the height and the area of the optical response measured, so 10 features in total. To be able to count the cells in a sample, the challenge is to distinguish them from all the noise and background. Currently the manual process to cluster the data consists in plotting 2 representative features (often B675-H and B530-H) and then drawing a polygon by hand around the points assumed to be cells, as shown in Figure 2. The number of cells is then approximated to be the number of points within the polygon.

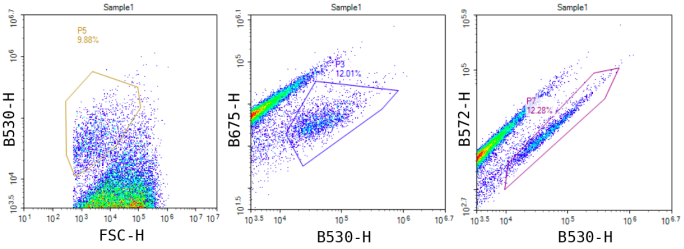


Figure 2: Hand drawn separation polygons

We see that this method includes several limitations: it combines only 2 features to make the separation, it assumes there are no background events near the cell events (within the polygon), and that no cell events are situated far from the other cell events (outside of the polygon). Further, the simple geometry of the drawn polygon sets constraints as to which points are considered as cells.

Mr Hannes Peter, a scientific at SBER, has thus turned to us to find, under his supervision, Machine Learning models and tools which are not subject to these limitations, to build accurate and easily exploitable clustering results.

3 Data processing

3.1 Data format

To import the FCS format files (specific format used to encode cytometry data) in our scripts, we used the FlowCal library. Those files contain, for each supposed cell in the

fluid, the three measured optic values of the laser beam, the forward-scatter light (FSC) and the side-scatter light (SSC).

As explained before each output is divided in a height and an area feature. Since they are highly correlated and their difference are not significant for our purpose, we chose to consider only the five height features (lower dimensions).

3.2 Data standardization and outlier removal

To process the data, we first import the file as mentioned above. The first step is to standardize the data. As the values increase exponentially, we first take their natural logarithm (all data plots in this paper are in logarithmic scale). Since the log function is undefined for negative values, they are removed from the data-set.

Then we standardize the data, by subtracting the mean to each file entry and then dividing by the standard deviation.

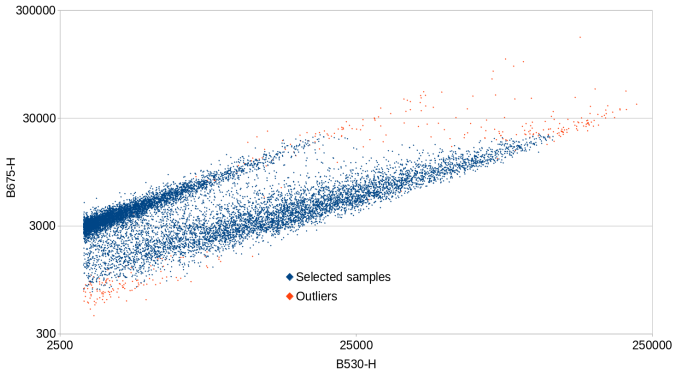


Figure 3: Example of outlier classification

The last step consists in removing the outliers, i.e. points with an abnormally high standard deviation. After our expert's validation, removing points that have a greater than 4 Euclidean distance to the mean leads to desired results.

4 Models

To cluster our data we implemented several algorithms leading to different models. For the sake of comparability, all result examples presented in this section are found by applying our algorithms to the same file ('GL10_UP_2B.fcs') considering the same channels ('FSC-H', 'SSC-H', 'B530-H', 'B572-H', 'B675-H'). For the sake of clarity the results are plotted only on two features: 'B675-H' and 'B530-H'.

4.1 K-means

A well known clustering algorithm is K-means. It takes as input the number of clusters it should divide the points in, and then randomly chooses that number of points to be the starting centroids. The algorithm progresses iteratively: at each step every point is assigned to the same cluster as its closest centroid, then every centroid's position is updated to be in the center of its cluster.

This algorithm may not find the most optimal solution where all points are at minimum distance from their cluster centroid, but if the metric used for calculating distances is the Euclidean distance, the convergence is guaranteed.

4.1.1 Parameters

The only k-mean parameter is the number of final clusters. As we are interested in splitting the data between the noise and the cells, and as this algorithm is not able to place the eventual outliers in a separated cluster, we remove the outliers beforehand, as explained in the previous section, and we fix this parameter to 2.

4.1.2 Results

This algorithm performed fairly well and gave us a good starting point.

The problem we run into is the lack of parameters. K-mean rigidity does not allow a wide range of different results. Also, data geometry is such that the distances to centroids do not well account for the noise and cell signals distribution.

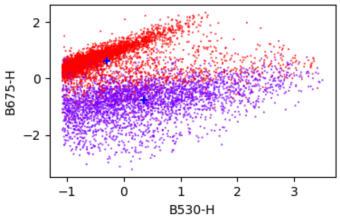


Figure 4: K-means clustering example, centroids are marked with blue crosses

4.2 Density

We also implemented a model based on the density of the data-set using DBSCAN (density-based spatial clustering of applications with noise). The algorithm takes as input a radius r and a number n which set a density threshold: a point is over the threshold (we say 'dense') if it has n or more other points in a r radius around it, and under ('sparse') if not. The global idea of DBSCAN is that it assigns the points that are below the threshold to a first cluster, and then it clusterizes the dense points to as many clusters as there are disjoint groups of points.

The way it does that is by identifying:

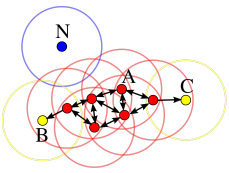


Figure 5: Points

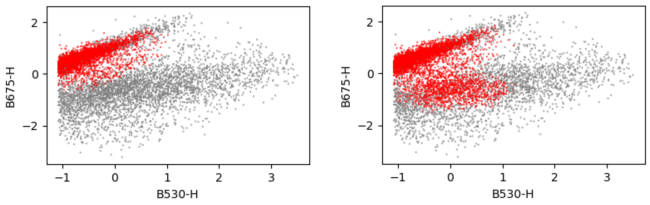
- **Core points**: dense as well as all neighbors (red points here)
- **Edge points**: sparse but have at least one core point neighbor (B, C)
- **Outlier points**: sparse and have no core point neighbor (N)

So outlier points will form one cluster, and every group of neighbor core and edge points form new clusters.

What we want to do to achieve our goal is to separate our cytometry data into three clusters: the sparse noise (an outlier cluster), the very dense background noise (a dense cluster), and the cells (another dense cluster).

4.2.1 Parameters

We should chose n and r so that the density threshold is low enough for cells not to be clustered as outliers, but high enough for the two dense clusters to be distinct (points between both clusters have to be classified as edges, not core points).



(a) $n = 225$ (b) $n = 187$

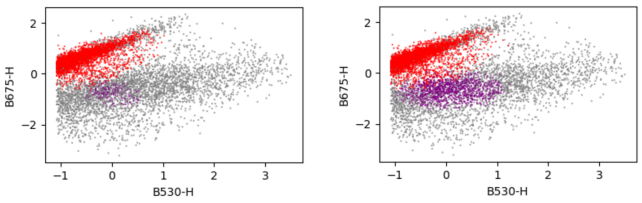
Figure 6: Tuning density parameter n with $r = 0.7$

We see in Figure 10 a an example of data where, with a radius $r = 0.7$, n has to be under *225* as otherwise density threshold is too high and cells are classified as outliers.

In Figure 10 b we see that for the same data, n has to be over *187* as otherwise density threshold is too low and there are too many points classified as dense between the big noise cluster and the cell cluster, therefore both clusters are not distinguished from each other and are classified as one.

4.2.2 Results

Tuned manually, this model gives interesting results.



(a) $n = 224$ (b) $n = 188$

Figure 7: Tuning density parameter n with $r = 0.7$

In Figure 7 we plotted the results for both extreme values of n (with r still to 0.7) for which our model finds three clusters. So now n can be chosen anywhere between *188* and *224* depending on the desired size of the cell cluster. We chose r to be 0.7 because after a lot of tests it seemed to be the value that would give us the largest window of n leading to three clusters.

Here the most interesting plot is this in Figure 7 b as it shows the clusters that contain the highest number of cell points (lowest density threshold) we could find with DBSCAN.

When asked, the expert in this domain said that even in our plots containing the highest number of cell points like Figure 7 b, as though the clusters seemed of the right shape, there were too few cells and too many outliers compared to reality. Since, as explained in 4.2.1, the range of threshold values for which the model outputs three clusters is very narrow, we were not able to tune this algorithm to greatly increase the size of the cell cluster.

4.3 Gaussian Mixture

Gaussian Mixture is a more advanced technique, in which each cluster is assumed to follow a different multidimensional Gaussian distribution. The clustering of a point is done by comparing the probabilities of it being generated by each Gaussian. As the Gaussian distribution is highest around its mean, this model is sensible to distances to centers and can therefore have similar results as K-means. In fact it can be summarize as a flavour of K-means where the clusters do not have to be circular (or hyper-spherical in higher dimensions).

We implemented this model using Scikit-Learn, more precisely the sklearn.mixture.GaussianMixture function.

4.3.1 Parameters

Scikit-Learn's GaussianMixture takes two inputs: the number Gaussian distributions (simply the number of clusters) and the covariance type. The covariance type is an indicator of the degree of freedom you want your cluster shapes to have. It can take the three different values illustrated in Figure 8: *diag*, *spherical* and *full*.

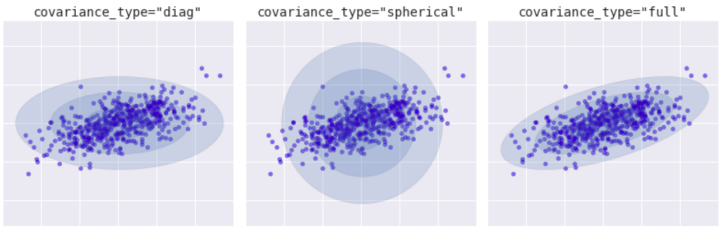


Figure 8: Covariance type ellipses

When set to *diag* (the default value), the distributions are constraint to be independent over each dimension but are free to be different. This results in an elliptic shape as shown in the left part of Figure 8.

When set to *spherical*, the distributions have to be independent and identical over each dimension which results in spherical shapes as shown in the center of Figure 8. In this case the GMM classification will be very similar to the K-means' as it reduces to a distance analysis.

In our case we set the covariance type to *full* which is more computationally expensive but sets no such constraints. As our features are highly correlated, we want our distributions to be shaped as ellipses of any orientation like on the right in Figure 8.

4.3.2 Results

This algorithm with inputs $n\_components = 2$ and $covariance\_type = full$ gives the most promising results, with a clear distinction between the two trails on Figure 9.

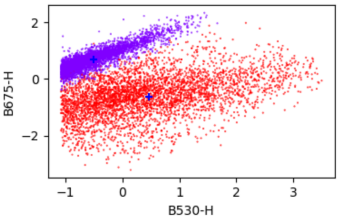


Figure 9: GMM (and final) clustering example

Mr Hannes Peter, the expert following this project indeed confirmed that these results correspond to his demand.

At this point we need to find a way to assign the correct label to each cluster. By comparing a lot of samples and files we notice that the cell cluster is generally placed higher on the x and lower on the y axis. So we tried computing the clusters centroids and simply comparing their $x - y$ difference, and assigning the "cell" label to the highest centroid. This works as desired for all our train and test files.

5 Combining models

GMM works well for our problem and the main background strand can be deleted meeting up to the expectations of the expert following the project. Once this inconvenient noise cluster is taken out of the picture, we notice, with the Mr Peter's feedback, that in reality there are usually less cells than what we have left which means some of the remaining points are still due to noise.

At this point and for the last step, we then use our density algorithm (presented in 4.2) on the product of GMM to extract the cells among the remaining noise.

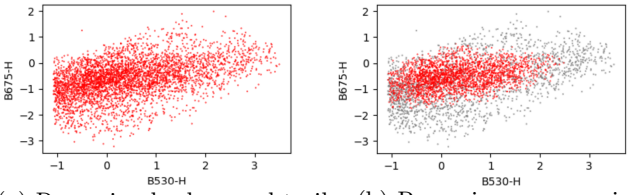
Parameters

The principal parameter here is the proportion of points to consider as cells, call this the *data_ratio*. Mr. Peter

offered to use some other independent method to count the cells in some of our samples for us to check our results, unfortunately due to the time constraints we were not able to obtain exploitable values. The idea would have been to compare our number of points with the number of cells in this independent counts, see if the ratio is similar over different files and find how to obtain it for a new file.

Since we can't do this, we will assume the *data_ratio* as a given input. Now we want to run a density algorithm that will split our data according to this *data_ratio*. Since density algorithm is computationally expensive and takes two input parameters (radius r and a number n), this is quite tricky and we proceed the following way: we fix a value for radius r and, by bisection, find number n for which density reaches $data\_ratio \pm trust\_interval$. If no such value is found, we increment r and start again.

Results



(a) Removing background trail (b) Removing sparse noise

Figure 10: Removing noise with $data\_ratio = 0.7$ and $trust\_interval = 0.1$

We reach our final results in this steps! By simply tuning the *data_ratio* we can (and we did) reach the final product corresponding to our problem specifications.

6 Detecting invalid sample

All the results presented here were shown for the 'GL10_UP_2B.fcs' file, which is quite representative of the general sample quality.

However, some files looked quite different and, even after going through processing and tuning, our results didn't seem to correspond to cells. When asked, our expert said that these files were simply invalid due to experimental weakness or errors unlinked to our model's performance, and were to be discarded.

In order to detect these samples for the laboratory to know which experiments would need to be run again, we have developed the following method that we apply on the GMM results (before the density selection). We first compute the distance and the slope between the cluster centroids then we detect two possible anomalies with the following code:

```
if distance < 0.55:
    print("!! Data is suspicious, the clusters are
    very close to each other, or not enough cells")
elif slope > 0.8:
    print("!! Data is suspicious, the algorithm
    detected only one cluster")
```

The parameters were found by observing a few files, and this method was then tested on all provided files and detected the exact files it was expected to.

7 Conclusion

Final Result

To summarize the steps we do on a file to obtain our best results: we first discard the area features, then standardize the data, remove outliers, apply GMM in *full* mode, check for invalid samples and finally apply density to discard remaining noise. Our final result for the example file presented in this paper is shown in Figure 11.

Figure 11: Final result, cells are in red

This result is very satisfying as, according to the expert, it indeed corresponds to the requirements and increases the counting accuracy. As a reminder, our product uses much more information (e.g. scatter and fluorescence measured in other channels) than what is currently done in bi-featured graphs. So, this makes much better use of the available data and since this is independent of the scientist, (particularly in a project where many scientists work together over many years), an objective algorithm will produce more replicable results.

Final Program

As we worked for the SBER Laboratory at EPFL, one of our main tasks was to provide a software implementing the algorithms described above. All our work was done in Python, and we created a program capable of applying any model to all files in a given directory.

This software can output a wide range of graphics, useful to check if the algorithm works correctly or if a sample is unusable (then flow cytometry has to be run again).

Future work

To push the fine tuning even further, we could, as explained in Section 5, find the data_ratio by using Mr. Peter's independent count.

Something else that could be interesting to do would be to use Scikit-Learn's predict_proba method that returns a matrix of size [n_samples, n_clusters], giving a mesure of the probability that any point belongs to the given cluster. We could use it to give certainty estimations or to weight each cell by it's probability to be a cell in the final cell count.

References

[1] *Fundamentals of Flow Cytometry.* https://www.aatbio.com/resources/assaywise/2019-8-1/fundamentals-of-flow-cytometry

[2] *A guide to gating in flow cytometry.* https://www.bio-rad-antibodies.com/blog/a-guide-to-gating-in-flow-cytometry.html