Cells and gene expression: a classification problem

Introduction

To study the link between epigenetics and memory, Giulia Santoni, a PhD at EPFL, decided to measure gene expression levels of neuron cells studied under three different conditions: “KAT5”, “eGFP” and “CBP”. Our machine learning project consisted in predicting, for a given cell, what the tested condition was in regard to what genes were expressed by this cell. For this classification problem, we came up with two final optimal models: one linear and one non-linear. In the following paragraphs, we will explain all the steps that led to these results.

1. Data treatment, PCA, and a look at clusters

Our training set, which contains the normalized counts for 32285 genes in 5000 different cells, was very large, and needed data cleaning before starting our modeling. We first removed all the constant and strictly correlated parameters (Data_treatment.jl, visualisation.ipynb) and managed to remove 6196 columns! The constant parameters were probably housekeeping genes, since they had the same expression level for all three categories. We started running a few models with this new treated data, but the number of parameters was still large, and running our machine extremely time consuming. So, we performed a PCA transformation on our data set to reduce its dimension (PCA.jl). We chose to preserve a variance ratio of 0.99 and thus not to compute all principal components, reducing even further the dimension of our data set. It also reduced overfitting problems by getting rid of some variability in our original data. The results obtained with this transformation will be discussed in the following paragraphs, but it is already worth noting that the running time was significantly lower and led us to continue on running the following models with this PCA data.

To have a deep understanding of our data, we decided to visualize it with some plotting. The first ones we did (PCA_2D.png, PCA_2D_sd.png), showed how important standardization of our data was. We were able to distinguish 6 different clusters in 2D. This number of clusters, which is higher than the number of categories, could be explained at a biological level. It may correspond to different cell types, or cell location, which are also related to gene expression level. Encouraged by these observations, we ran a K-mean clustering algorithm (clusters.jl). However, after a few trials with different ‘k’ (we thought 3 to 12 was a good range, because there were 3 categories, and 6 identifiable clusters on the plots) and different random seeds, we failed to obtain a decent clustering. The resulting confusion matrices showed that either 1 or 2 categories were omitted by the machine which is why we stopped investigating clustering… Then, we looked at the PCA components. The biplot of the 1st principal component vs the 2nd one (Biplot.png), was hard to interpret as there were many parameters and all the principal components should be analyzed! We thus rather looked at how the variance was represented by our PCA (Pvar_explained.png). The first plot shows the proportion of variance explained versus the number of components. We can notice a big change in the slope near the 200th component and therefore conclude that the first 200 components explain a more important proportion of variance. This is interesting information because we could have chosen to run our models using only these 200 first dimensions and get faster results. We decided not to, keeping more information for accuracy of our predictions to remain higher.

2. Linear methods and results

We started by running the simplest model: a logistic classifier (Regularization.jl). The predictions we got by submitting it on Kaggle was 86,4%. This good result incited us to further investigate this model. Noting that we got the message ‘proximal gradient descent didn’t converge’, we ran the model with PCA treated data. Indeed, the direction of fastest ascent becomes clearer as the dimensions shrink. We were therefore able to obtain better predictions (87,3%) 4 times faster. We then added a lasso regularization, which by maximizing

the weight of the most important genes and minimizing the weight of the least important ones, reduces flexibility and most likely gives better results when the machine is applied to a new data set. The lambda hyperparameter was chosen using a self-tuning machine. Rerunning the tuning when the best hyperparameter was a boundary value, we ended up with an optimal lambda of 4,6415e-7 and a Kaggle prediction of 90,776%. Additionally, we tried running a ridge regularization but as a simple model (without tuning) took several hours to run, we decided to stick to lasso regularization. Lasso also seemed more appropriate for our data set as it is able to drop some predictors.

In another file (KNNandPN.jl), we introduced a polynomial model of which we tuned the degree. But the predictions were only 37,8%, we concluded that the degree 1 (Logistic classifier) was the best and we moved on. Finally, we coded a KNNClassifier, but just as for the ridge regularization, we gave up because of time constraints and chose the tuned logistic classifier with PCA and L1 regularization as our final linear model.

3. Non-linear methods and results

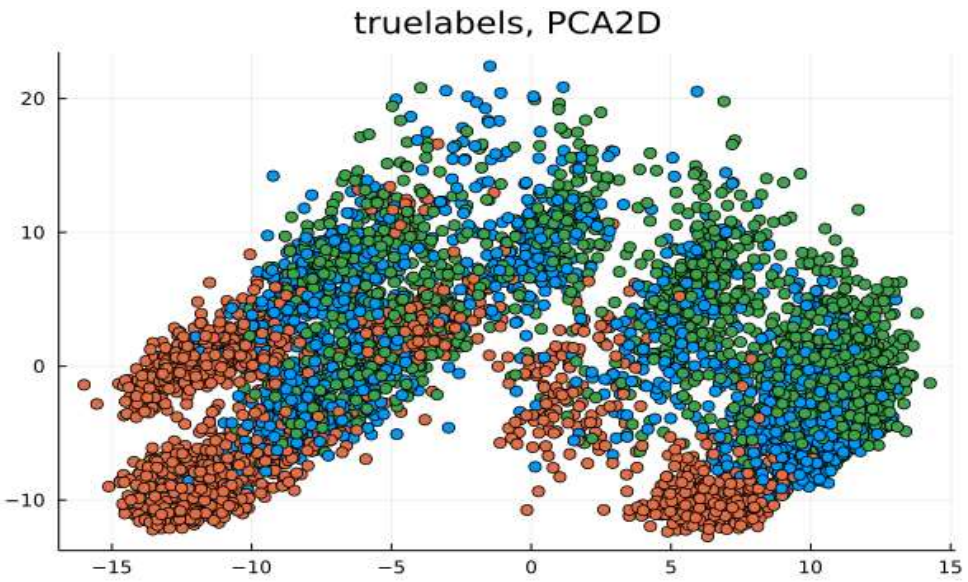
For our non-linear modeling (non_linear_methods.jl), we first decided to try to see how a Random Forests machine would perform. We started with a simple model and tried to see how the number of trees impacted predictions. We tried with 300, 750 and 3000 trees and for all these trials, we were only able to score a maximum of 67% accurate predictions on the Kaggle test set. Optimizing other hyperparameters and running a Gradient Boosting Trees did not seem a priority for us as the results were already too low. We rather preferred focusing on building a strong and efficient neural network. To do so, we first looked at a simple neural network and tried to see how different parameters and hyperparameters would influence the classifier. For all these tests, the prediction accuracy on test sets were above 80%. We also noticed that a L1 regularization yields faster results and good predictions. With all these observations, we started building an optimal neuron network classifier. As we have a large dataset and multiple hyperparameters, it was impossible to perform a cross-validation on a self-tuning machine. Based on our preliminary observations, we chose to use the PCA treated data and a lasso regularization with a lambda of the same range as for the optimal logistic classifier. The number of layers and neurons was set arbitrarily but based on the previous observations. We however decided to tune the epochs and dropout hyperparameters. Tuning the epochs was important because we knew we needed a higher value but didn't want the running time to skyrocket. Therefore, we plotted a learning curve with a maximum number of iterations at 1'000 (learning_curve_NC_L1_PCA_epochs4.png). We chose not to go higher to avoid having a time-consuming machine. Setting the epochs at the most efficient value from the learning curve, 800, we then tuned the dropout. Tuning this hyperparameter is important to avoid overfitting while retaining the important components. We separated our train data in subsets of 3000 and 2000 cells, and computed the confusion matrices for dropouts of 0.01, 0.1 and 0.3. The final model ran on the whole training dataset with epochs of 800 and the dropout value that gave the smallest misclassification rate in the confusion matrices: 0.1. This is not the model that gave the best result on Kaggle's public leaderboard, but it is the one that is least likely to overfit the 20% of the test data used for these predictions.

Conclusion

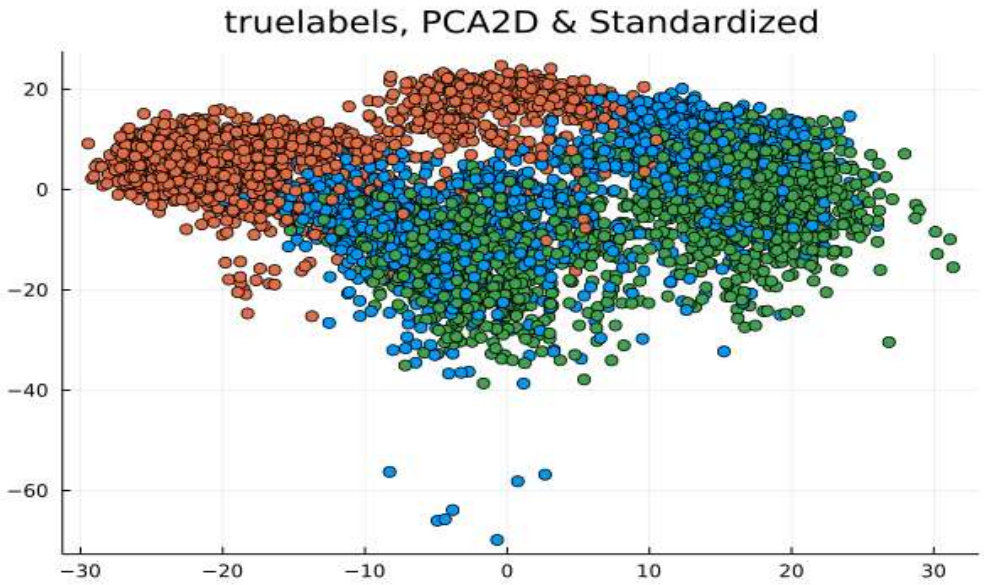
Having a big dataset, this project encouraged us to explore various data cleaning possibilities and make decisions that would optimize the running time and minimize the overfitting while minimizing the loss of information. In the end, the optimal linear method we chose is a logistic classifier with a tuned L1-regularization that is trained with a PCA transformation of our data. As for the non-linear model, we chose to submit a model that doesn't have the best prediction of the public leaderboard but whose hyperparameters have been evaluated to avoid overfitting. Overall and as of the deadline time, the best model seems to be the final linear one.

Annexes

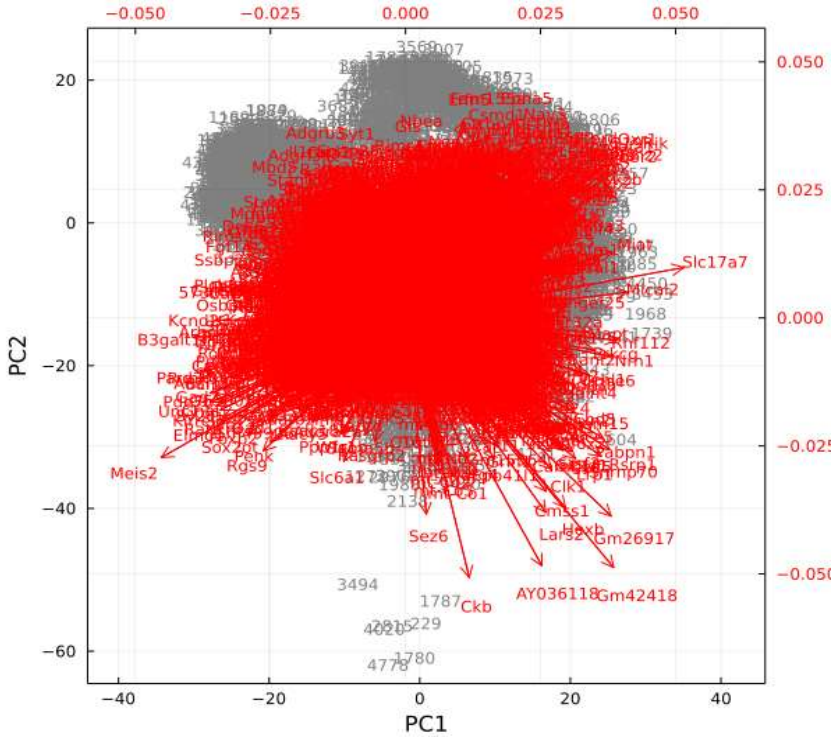
PCA_2D.png



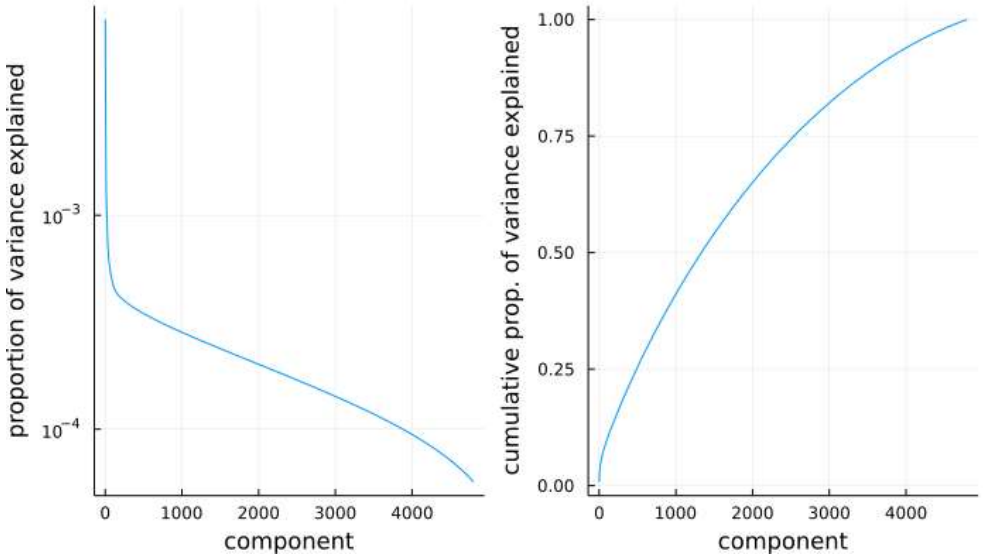
PCA_2D_sd.png



Biplot.png



Pvar_explained.png



learning_curve_NC_L1_PCA_epochs4.png