BioE245 Final project

Hugo Hakem Meng BioE 23-24'

Acknowledgement

This work stands as the final project for the BioE245 class taught in Spring2024 by Professor. Liana Lareau, and GSIs. Michael Herschl, Amy Lu.

****Disclaimer****

This small report is meant to present my results in a concise way. For more details on the methodology used, please refer to the Notebook part 1 and part 2 as they have been specially designed to explain my methods or motivations.

Part 1: Lower-dimension representations of the cells

One AutoEncoder architecture has been implemented and fine-tuned (in term of l1, l2 regularization see **Annex Figure1** to better understand) so the cluster in the PCA lower dimensional space fed with the embedded features display descent cluster. For more details on the method used or on the AutoEncoder architecture itself, please refer to the Notebook Part1. In **Figure1** a plot comparing the different dimensionality reduction method.

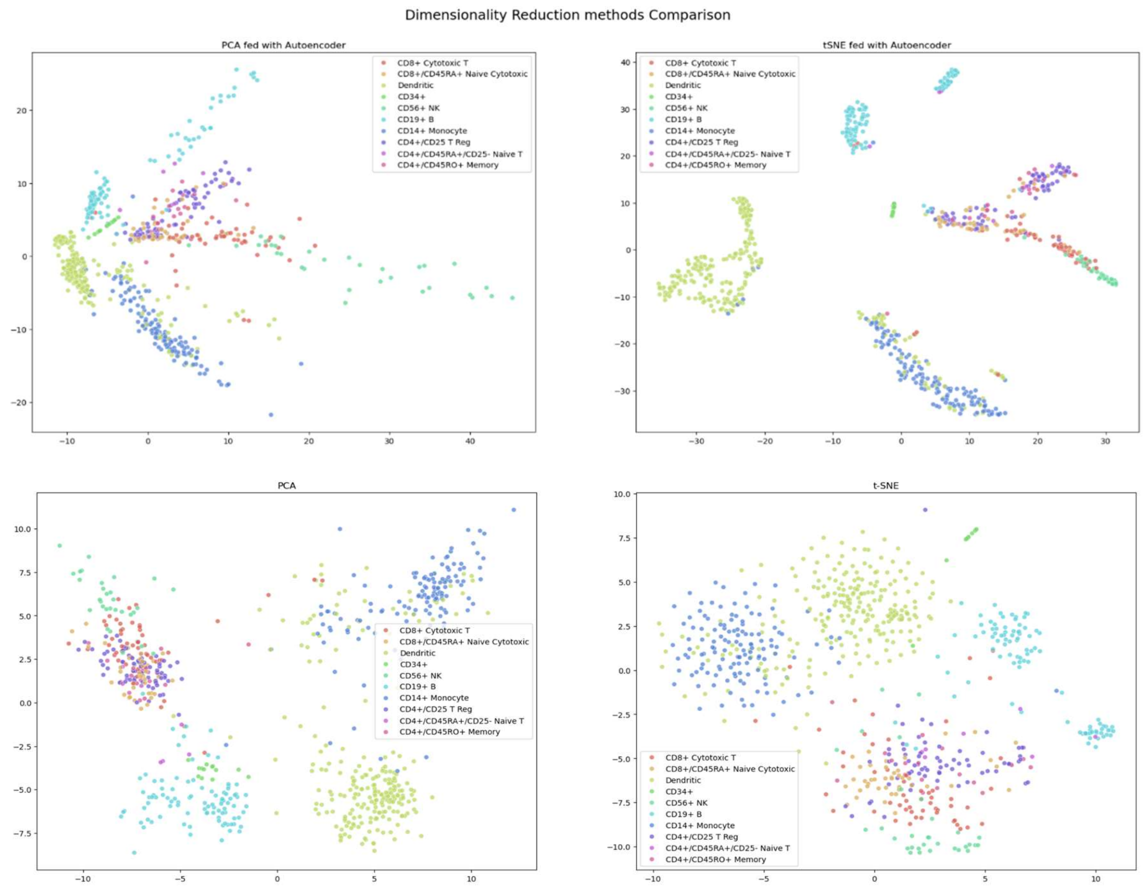


Figure1: Dimension reduction methods comparison

Upper left: PCA fed with AutoEncoder / Upper right: tSNE fed with AutoEncoder

Bottom left: PCA only / Bottom right: tSNE only

When comparing the result of each techniques, several conclusion can be made:

- **None of the plot are essentially better** than the other. Indeed, none of them give a clear separation for each class.
- Both **PCA and tSNE has the interest to give a very deterministic response**. While the AutoEncoder may have to be run several times with the exact same parameter before to reach a descent clustering.
- Looking at **PCA or tSNE fed with autoencoder, both of them give tighter cluster than PCA and tSNE only** and ultimately a better separation of each class.
- tSNE is the plot which gives the worse cluster in term of density. The reason is that tSNE is known to perform bad when feed with a large number of features (I refer to the tSNE paper). A better practice is to feed it with PCA or with autoencoder as before.

The question remains, which technique to choose. My answer will depend on what is the purpose.

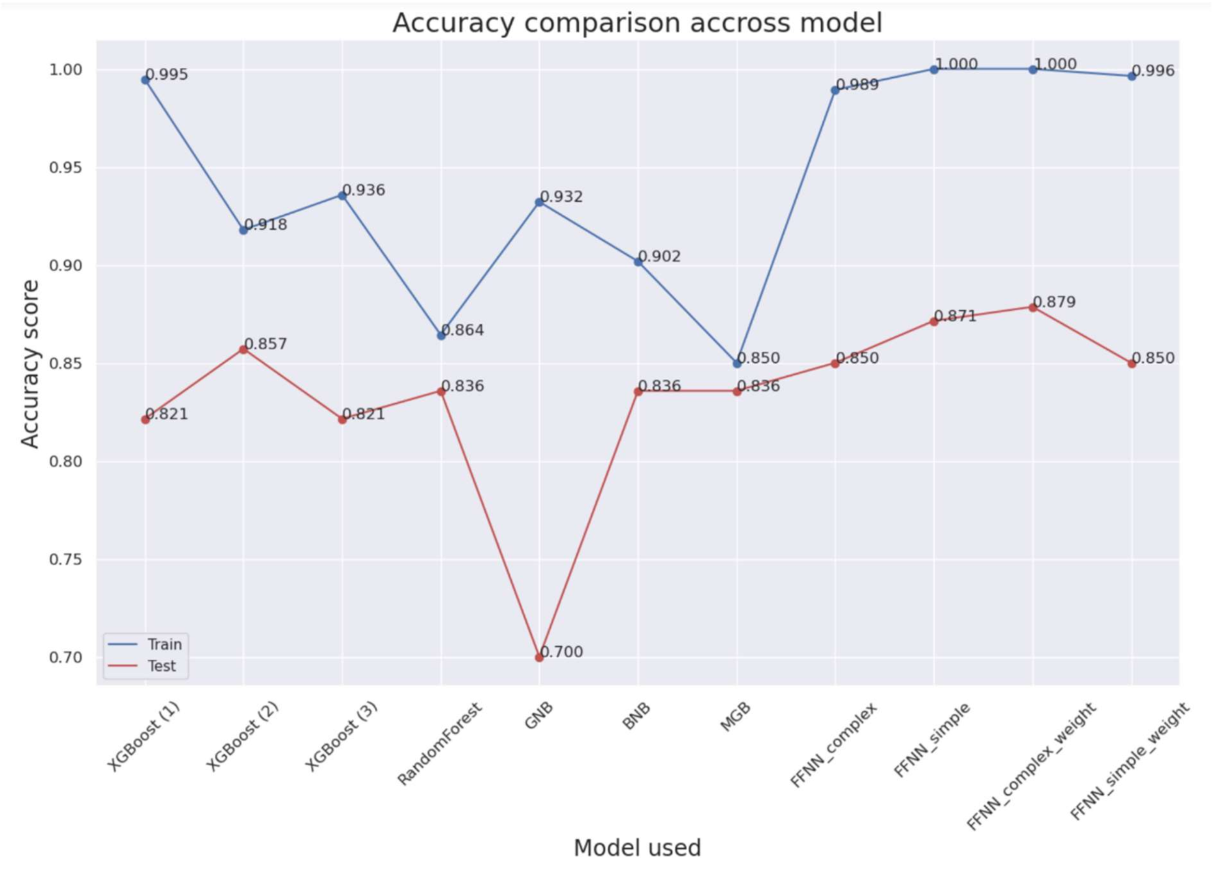
- For **Unsupervised Learning**:
    - Here a clustering algorithm will be used. The more separate the cluster are, the better. I would therefore go for PCA or tSNE fed with autoencoder.
    - In particular, for an algorithm such as a **Gaussian mixture model**, I would prefer the **PCA fed with autoencoder** as the **cluster are more elliptic**.
    - For an algorithm such as **DBSCAN**, which is based on the **cluster density**, I would prefer **tSNE fed with autoencoder** as it would reduce the number of cells classified as outliers.
- For **Dimension reduction for Supervised Learning**:
    - Using **PCA provide a systematic method to apply to all data set**. The consistency of PCA is preferable over the autoencoder as it response may vary a lot. And tSNE should not be used as it doesn't perform well on large amount of input feature.
    - However, in the context of supervised learning, there is no need to create an autoencoder, to reduce the dimension input and then classify using a Feed Forward Neural Network (FFNN). The process is repetitive, and a classic FFNN reducting directly the input feature data to a class prediction is preferable as the model tuning will be directly quantified in term of accuracy performance, rather than having to tune both an autoencoder and a FFNN.

Part 2: Classifier

For this part, several classifiers have been tested. Again, for more information on their implementation please refer to the notebook part2.

1. **Decision Trees Algorithm**
   Why? They are easy to fine-tune, and they can lead to high accuracy in classification task.
   - **XGBoost (3 set of parameters tested)**
     XGBoost is an optimized implementation of a Gradient Boosting Decision Trees (GBDT). GBDT is usually the most powerful decision trees algorithm as it is specifically designed to reclassify wrongly classify data. However, it may lead to overfitting on the training set.
     For the 3 set of parameters, those are attempted to reduce overfitting.
   - **RandomForest**
     Observing that XGBoost is overfitting, Random Forest is a common decision trees algorithm to address these issues. Adverserly, it may lead to underfitting.
2. **Bayes Model**
   Why? Again for simplicity purpose, Bayes model are used as no fine-tuning is recquired and as they may lead to fairly good accuracy depending on the data set.
   - **Gaussian Naive Bayes (GNB)**
     Commonly used for continuous features. Gaussian Naives Bayes use a gaussian posterior probability and assume independency of features.
   - **Bernouilli Naive Bayes (BNB)**
     Commonly used for binary features. Bernouilli Naives Bayes use a Bernoulli posterior probability and assume independency of features. The problem is that the features we are working with are continuous. Unconsciously, the Bernouilli Naives Bayes has been applied and still work. I don't have a perfect reason why the classification could still happen. My assumption is that the posterior probability is actually compute following this formula:
     $$P(x_i|y) = P(x_i = x|y) * is(x_i = x) + P(x_i \neq x|y) * is(x_i \neq x)$$
     Therefore, the fact that this classifier work may be explained because the feature has a certain degree of discreteness in a sense that some features may have similar value across cells for a specific class.
   - **Multivariate Gaussian Bayes Model fed with PCA (MGB)**
     Used for continuous features. Multivariate Gaussian Bayes use a gaussian mixture posterior probability and **does not assume features' independence**.
     It has been fed with PCA as it happens to give better result.
3. **Feed Forward Neural Network (FFNN) (4 set of parameters tested)**
   Why? A FFNN is used as neural network offer more freedom in term of architecture and as they can learn nonlinear pattern in the data. The underlying problem with FFNN is that their fine-tuning may be very tricky as several layers can be considered or regularization technique etc. Here for simplification purpose, no scheduler for the learning parameter will be implemented.
   For the 4 set of parameters, those are attempted to reduce overfitting: Either by considering weight for each class as the data are imbalanced, or by removing hidden layers.
   See **Annex Figure 2** to see the training vs test loss / accuracy across epochs.

In **Figure 2**, a plot summarizing the different accuracy obtained for each model.

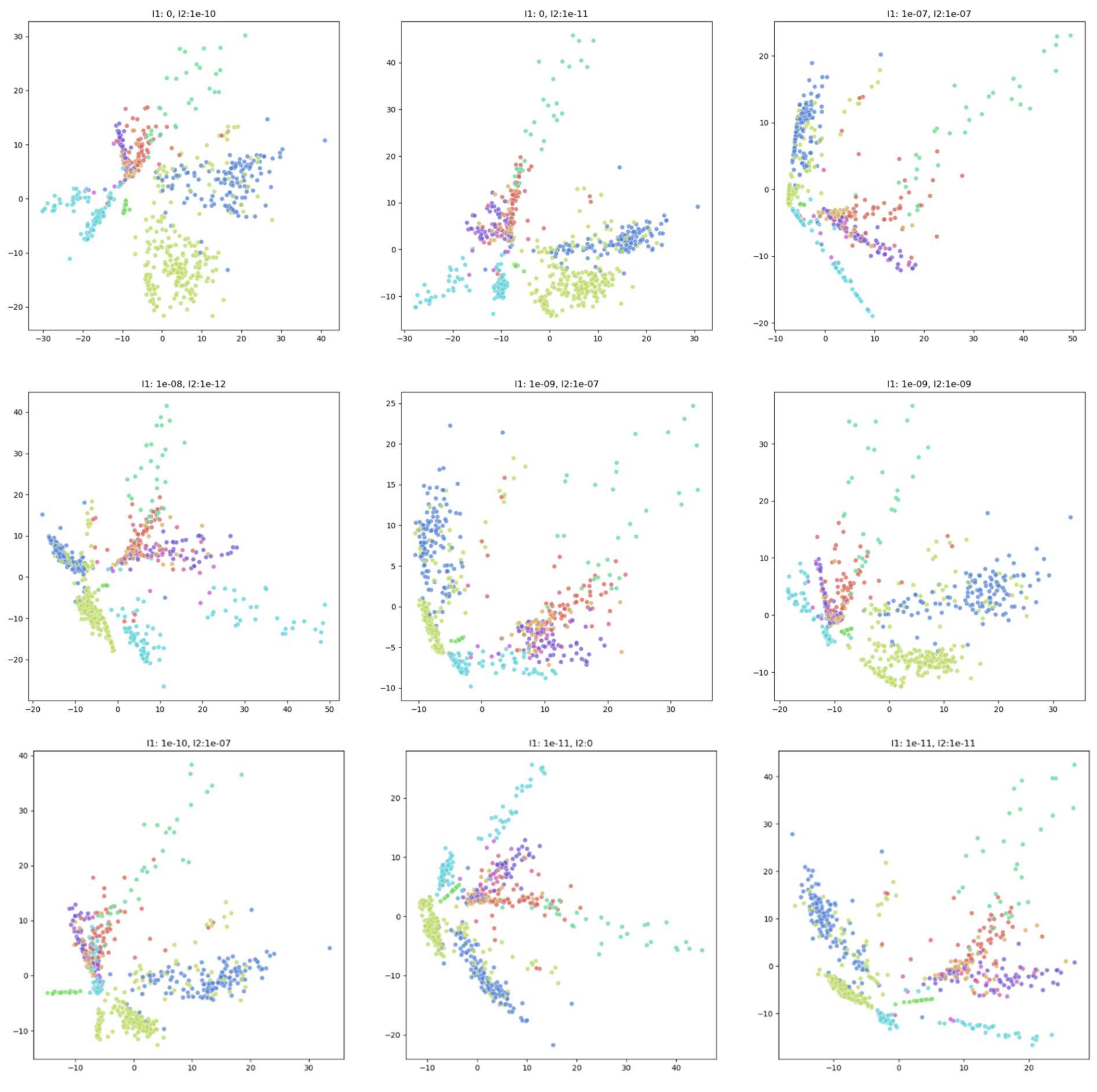


When looking at this graph several conclusions can be made:

- It happened that the **best test accuracy** obtained is with a **Feed Forward Neural Network (FFNN)** and in particular with the variant with a complex architecture and weight. However, FFNN are not deterministic, and their accuracy may vary across test from ~0.83-0.90. FFNN is however the most difficult to fine-tune and lead to a lot of over-fitting.
- **XGBoost** has the advantage to provide a more deterministic solution (across every test conducted, always the same accuracy for a chosen architecture). In particular the architecture used in XGBoost (2) provide a good trade off in term of Test accuracy and over-fitting.
- The **minimum over-fitting** is observed for the **RandomForest** Classifier and the **Multivariate Gaussian Model (MGB) fed with PCA**. MGB present the advantage to be completely deterministic, when random forest may reach various different accuracy from ~0.80-0.84. However, MGB had to be fed with PCA. Therefore, for larger data set with a bigger number of features (there is usually 24,000 genes instead of only 750 here), this method may not be as powerful.
- For **Naive Bayes method**, the gaussian one is not good enough, as it overfit too much. Bernouilli is unexpectedly good. It should however be avoided for larger dataset as X features may not be as discrete as in our case.

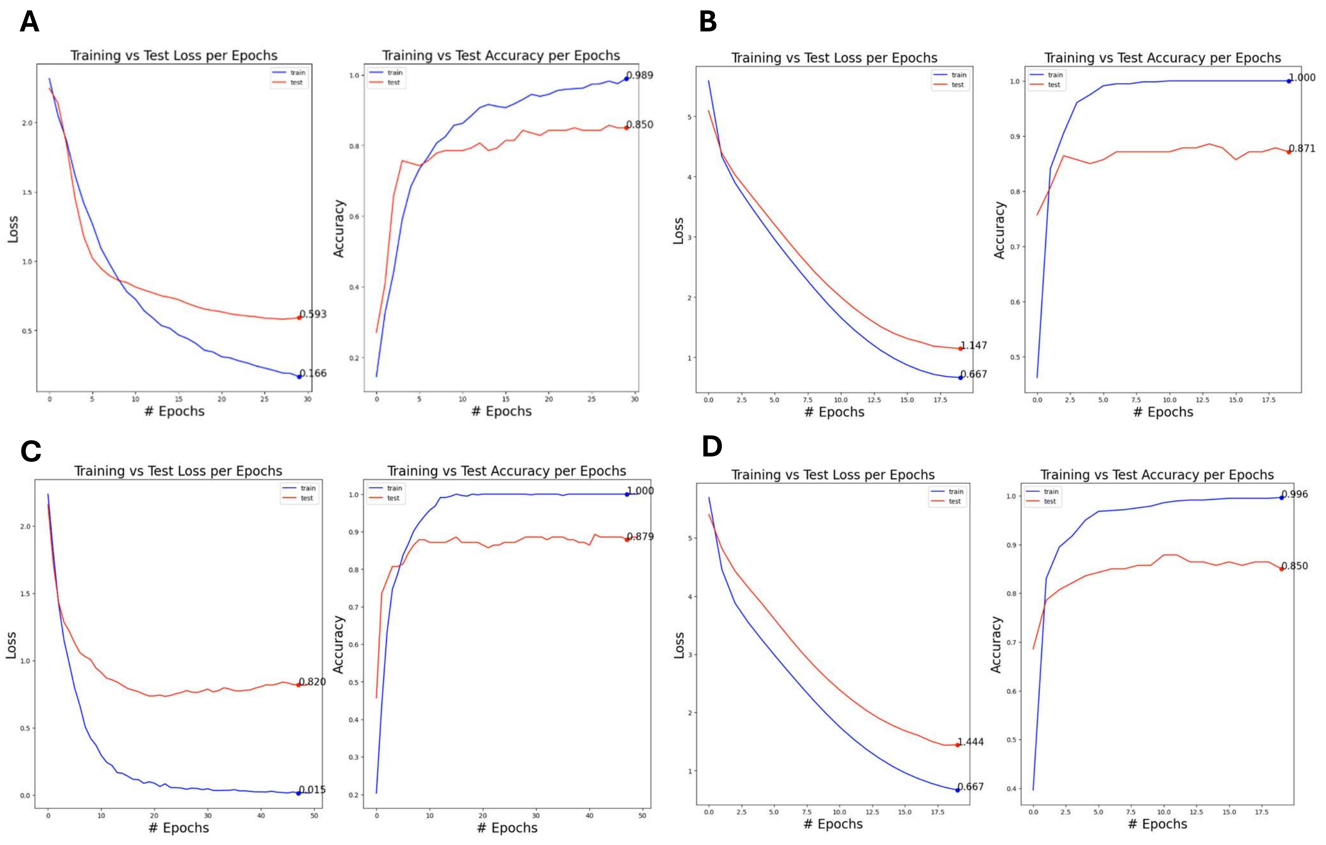
To conclude on the accuracy comparison for each model, FFNN is the most powerful, Decision Trees gives a fairly good result with a minimum amount of fine tuning, Multivariate Gaussian Bayes fed with PCA is the simplest and lead to still good result.

Supplemental Features for Part1 and Part2



Annex Figure1: Plot comparison of "PCA fed with autoencoder" with different regularization l1,l2 parameter.

All of the solution proposed are pretty good in term of clustering. For each penalty, some class seems very difficult to separate. Without a clear measure of the clustering efficiency, the penalty **l1=1e-11** and **l2 = 0** looks slightly better (that may be subjective) and is therefore the one chosen to compare with PCA only and tSNE only in the main body of this report.



Annex Figure2: Plot comparison of FFNN Training vs Test of Loss and Accuracy for different set of parameters. The dot for each line plot refers to early stop of the training.

A: Complex FFNN, **B:** Simple FFNN, **C:** Complex FFNN with weight, **D** Simple FFNN with weight

Notation:

1. **Complex**, refer to the complexity of the FFNN: this is simply the number of linear layers used. Here an architecture is said to be complex when **6 linear layers** are used vs **Simple**, when only **2 linear layers** are used.

2. **With weight**, means that the imbalance of the class has been considered. Weight has therefore been passed into the Cross Entropy function.

Annex Figure 2 display different training result for the FFNN architecture with 4 different set of parameters. Those set of parameters were used to try to address over-fitting or try to reach higher Test accuracy. If in term of test accuracy, the best one is **C:** Complex FFNN with weight, the result may differ as FFNN is not a deterministic method. To fully assess the accuracy of each method, different generation of train / test split should be tested, and each model should be tested several times.