Written Report – 6.419x Module 2

Name: (Inés da Rosa_inesdarosa)

Problem 2: Larger unlabeled subset

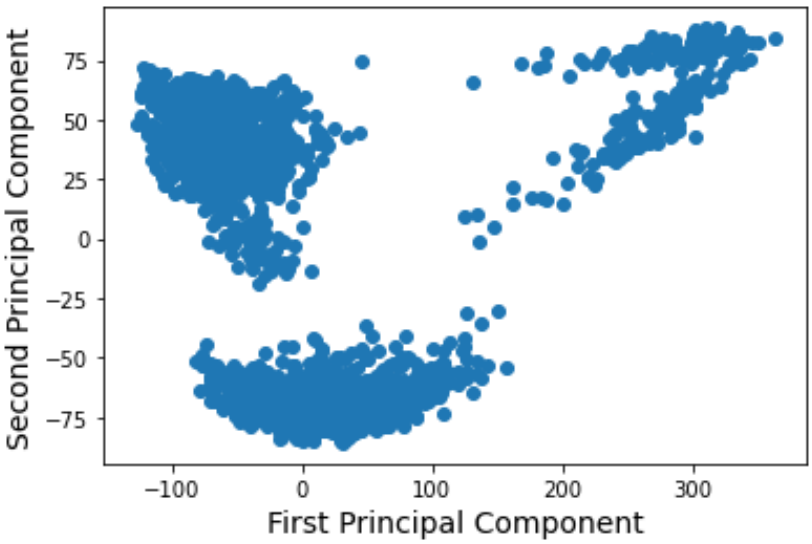
Part 1: Visualization

A scientist tells you that cells in the brain are either excitatory neurons, inhibitory neurons, or non-neuronal cells. Cells from each of these three groups serve different functions within the brain. Within each of these three types, there are numerous distinct sub-types that a cell can be, and sub-types of the same larger class can serve similar functions. Your goal is to produce visualizations which show how the scientist's knowledge reflects in the data.

As in Problem 1, we recommend using PCA before running T-SNE or clustering algorithms, for quality and computational reasons.

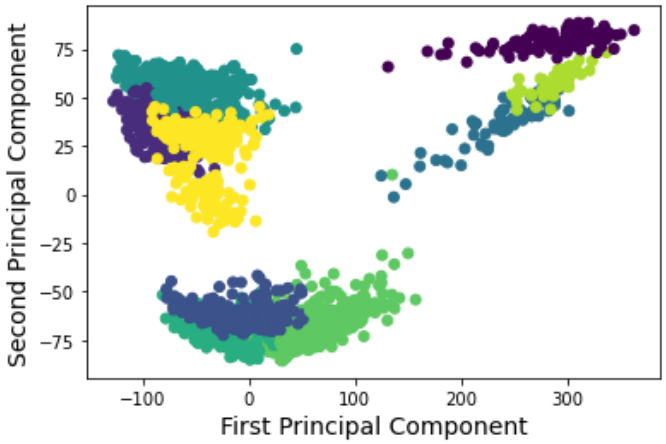
1. *(**3 points**) Provide at least one visualization which clearly shows the existence of the three main brain cell types described by the scientist, and explain how it shows this. Your visualization should support the idea that cells from a different group (for example, excitatory vs inhibitory) can differ greatly.*

   This is a PCA with two dimensions to represent the visualization. In the picture it is possible to see three groups, each of these groups represents each group of the three cell types (i.e. excitatory neurons, inhibitory neurons, and non-neuronal cells). The largest variance of the data is captured in the first component and the second largest variance is captured in the second component.



2. *(**4 points**) Provide at least one visualization which supports the claim that within each of the three types, there are numerous possible sub-types for a cell. In your visualization, highlight which of the three main types these sub-types belong to. Again, explain how your visualization supports the claim.*

We can visualize through the colors the subtypes in each principal groups. This grouping was performed by k-means cluster taken the data from the PCA components.

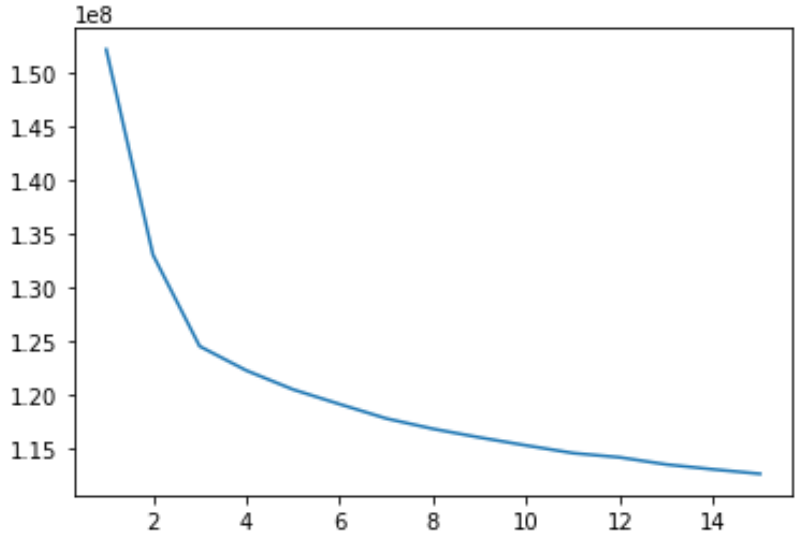


Part 2: Unsupervised Feature Selection

Now we attempt to find informative genes which can help us differentiate between cells, using only unlabeled data. A genomics researcher would use specialized, domain-specific tools to select these genes. We will instead take a general approach using logistic regression in conjunction with clustering. Briefly speaking, we will use the p2_unsupervised dataset to cluster the data. Treating those cluster labels as ground truth, we will fit a logistic regression model and use its coefficients to select features. Finally, to evaluate the quality of these features, we will fit another logistic regression model on the training set in p2_evaluation, and run it on the test set in the same folder.

1. *(**4 points**) Using your clustering method(s) of choice, find a suitable clustering for the cells. Briefly explain how you chose the number of clusters by appropriate visualizations and/or numerical findings.*

Through the Elbow method we can see the decrease of WGSS (Within group sum of squares) with the increase of cluster number. The data used was the values obtained from the PCA components. In this case the principal change happened in three cluster. Also, we can observe another step of change, less sharped, from three to nine clusters. This coincides with the cellular subgroup inside the principal cellular groups. For the next analyses is used the option with 3 clusters.



2. *(**6 points**) We will now treat your cluster assignments as labels for supervised learning. Fit a logistic regression model to the original data (not principal components), with your clustering as the target labels. Since the data is high-dimensional, make sure to regularize your model using your choice of , , or elastic net, and separate the data into training and validation or use cross-validation to select your model. Report your choice of regularization parameter and validation performance.*

The regularization parameter is 'l2', the scores tell me the values of the performance.

log_reg =
LogisticRegression(penalty='l2',C=10,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.951117

Multi-class logistic regression: *When the underlying data has more than two classes involved, we can adapt Logistic Regression which is usually used for binary classification by **one-versus-rest** approach. In particular, if we have classes, we train separate binary classification models using logistic regression. Each classifier for is trained to determine the probability of a data*

point belonging to class . To predict the class for a new point , we run all classifiers on and choose the class with the highest probability, i.e.,

Python tip:

You may use liblinear solver LogisticRegression or LogisticRegressionCV for one-versus-rest logistic regression.

Note:

Recall that the p2_unsupervised_reduced and p2_evaluation_reduced folders contain datasets with a reduced number of genes, in case you are unable to run some of the procedures on the larger versions. In particular, a full logistic regression could take 1 or 2 GB of memory to run.

3. *(**9 points**) Select the features with the top 100 corresponding coefficient values (since this is a multi-class model, you can rank the coefficients using the maximum absolute value over classes, or the sum of absolute values). Take the evaluation training data in p2_evaluation and use a subset of the genes consisting of the features you selected. Train a logistic regression classifier on this training data, and evaluate its performance on the evaluation test data. Report your score. (Don't forget to take the log transform before training an testing.)*

   Compare the obtained score with two baselines: random features (take a random selection of 100 genes), and high-variance features (take the 100 genes with highest variance). Finally, compare the variances of the features you selected with the highest variance features by plotting a histogram of the variances of features selected by both methods.

   I selected the 100 features following these steps:

   log_reg.coef_

   reg_coef = np.sum(np.abs(log_reg.coef_),axis=0)

   coef_sel = heapq.nlargest(100, range(len(reg_coef)), reg_coef.take)

   This is the log_reg with the 100 genes selected with higher coefficient:

   - log_reg = LogisticRegression(penalty='l2',C=10,solver='liblinear',max_iter=5000,

   multi_class='ovr').fit(x_ev_train_trans[:,coef_sel],y_ev_train)

log_reg.score(x_ev_train_trans[:,coef_sel],y_ev_train): 1.0

log_reg.score(X_ev_test[:,coef_sel],y_ev_test): 0.4711191335740072

This is the log_reg with the 100 genes random selected:

- log_reg = LogisticRegression(penalty='l2',C=10,solver='liblinear',max_iter=5000, multi_class='ovr').fit(x_ev_train_trans[:,rand_list],y_ev_train)

log_reg.score(x_ev_train_trans[:,rand_list],y_ev_train): 0.9870009

log_reg.score(X_ev_test[:,rand_list],y_ev_test): 0.240974729

We can observe the less performance with random data than the 100 genes selected with higher coefficients. Although the performance on x_test and y_test with the 100 selected genes is also low.

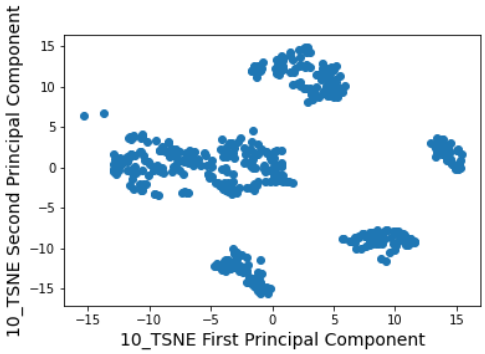
> ***Note:*** *The histogram should show the distribution of the variances of features selected by both methods. You could show the comparison by overlaying both histograms in the same plot.*
>
> ***Hint:*** *Refer to the recitation for some guidance if necessary.*

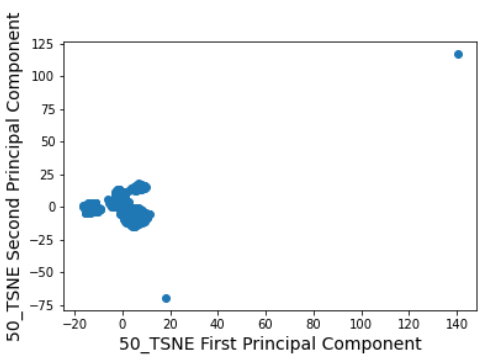
Problem 3: Influence of Hyper-parameters (Written Report)

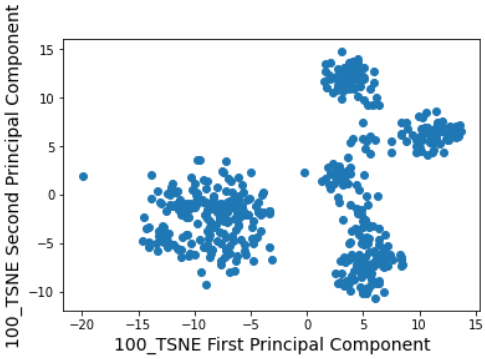
The hyper-parameter choices used in data analysis techniques can have a large impact on the inferences made. As you may have encountered, finding the best choice of parameter such as perplexity in T-SNE or the number of clusters can be an ambiguous problem. We will now investigate the sensitivity of your results to changes in these hyper-parameters, with the goal of understanding how your conclusions may vary depending on these choices.

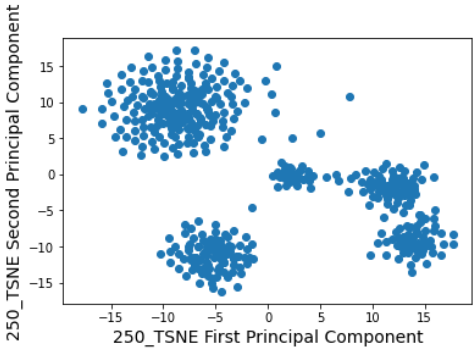
1. *(3 points) When we created the T-SNE plot in Problem 1, we ran T-SNE on the top 50 PC's of the data. But we could have easily chosen a different number of PC's to represent the data. Run T-SNE using 10, 50, 100, 250, and 500 PC's, and plot the resulting visualization for each. What do you observe as you increase the number of PC's used?*

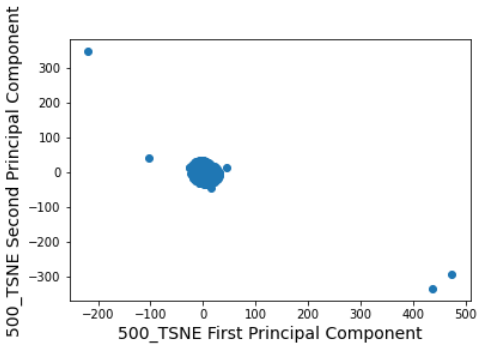
When the number of PC's increases it seems that the group structure disappears. For example, with 10 PC components we can observe groups but with 500 PC components it is not possible to distinguish those groups.











2. *(13 points) Pick three hyper-parameters below and analyze how changing the hyper-parameters affect the conclusions that can be drawn from the data. Please*

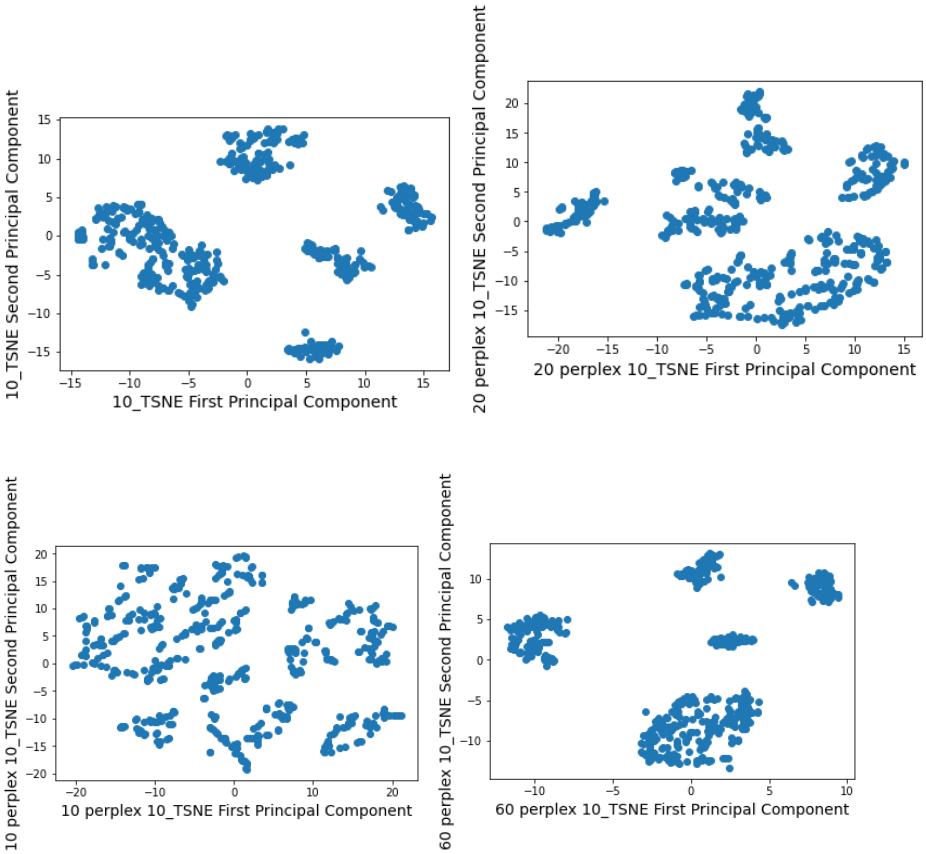
choose at least one hyper-parameter from each of the two categories (visualization and clustering/feature selection). At minimum, evaluate the hyper-parameters individually, but you may also evaluate how joint changes in the hyper-parameters affect the results. You may use any of the datasets we have given you in this project. For visualization hyper-parameters, you may find it productive to augment your analysis with experiments on synthetic data, though we request that you use real data in at least one demonstration.

Some possible choices of hyper-parameters are:

TSNE with 10 PC components: it is varied the perplexity and the iteration numbers.

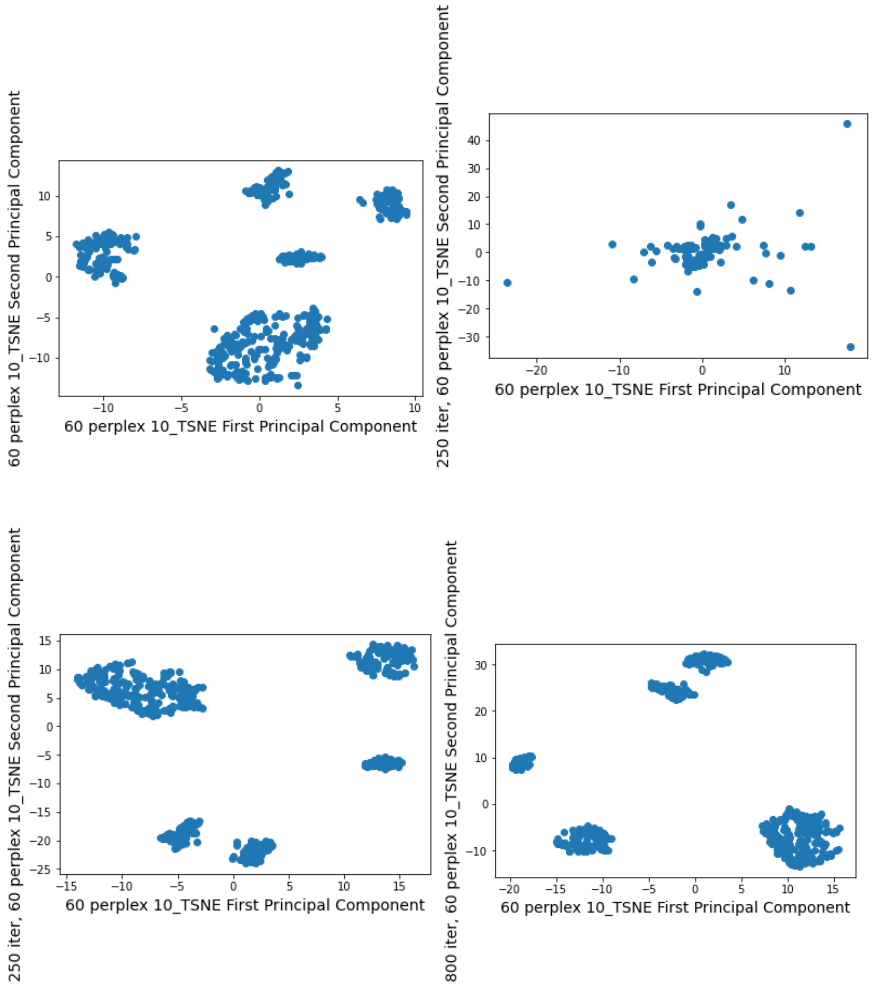
Iter=300

Perplexity=40 (upper left), 20(upper right), 10(lower left), 60(lower right)



Perplexity=60

Iter: 300(upper left), 250(upper right), 500(lower left), 800(lower right)



It is possible to observe that the perplexity with number 60 represents a structure with groups well defined. It possible to perceive an improvement of the structure with groups well defined with the increasing of number iterations.

Here it is modified the regularization parameters l1 and l2 in the log_reg. It is showed that the best performance happened with l2 and C=10. C is inverse of regularization parameter, it also was variated. The values of C were selected considering the values that appear in the recitation.

- log_reg = LogisticRegression(penalty='l1',C=0.1,solver='liblinear',max_iter=5000, multi_class='ovr').fit(X_train,y_train)

  log_reg.score(X_train,y_train): 1.0
  log_reg.score(X_test,y_test): 0.89804469

- log_reg = LogisticRegression(penalty='l2',C=0.1,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.92178770

- log_reg = LogisticRegression(penalty='l1',C=1,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.92597

- log_reg = LogisticRegression(penalty='l2',C=1,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.9497206

- log_reg = LogisticRegression(penalty='l1',C=10,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.9329608

- **log_reg =**
LogisticRegression(penalty='l2',C=10,solver='liblinear',max_iter=5000,
multi_class='ovr').fit(X_train,y_train)

log_reg.score(X_train,y_train): 1.0
log_reg.score(X_test,y_test): 0.951117