Written Report – 6.419x Module 2

Seokyu Kim

March 2023

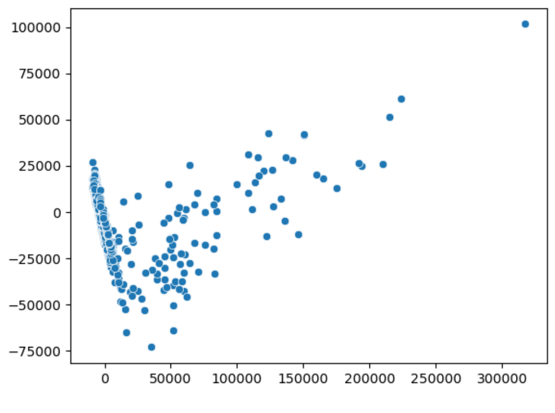
Problem 2: Larger unlabeled subset (Written Report)

Part 1: Visualization

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

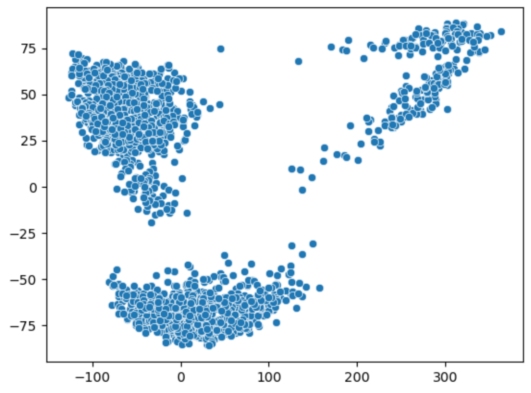
Solution: We will determine the number of main brain cell types using a Principle Component Analysis (PCA) graph. First, we make a PCA plot of the raw data and visualize before applying any specialized techniques. We observe the following plot in Figure 1:

Figure 1: Plot of PCA's using Raw Data



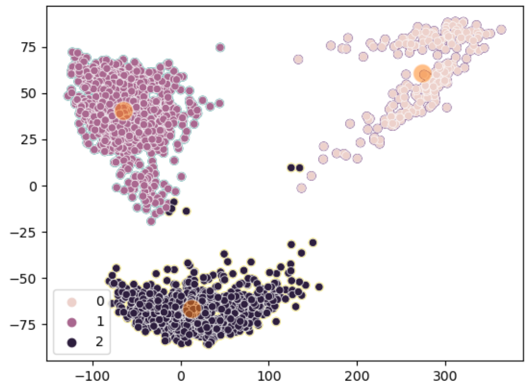
We do not see any distinct groups of data points. This is likely because there are genes with extremely high expression values in a very few number of cells. We can resolve this issue by scaling the data set with a logarithmic transformation. Figure 2 provides the PCA plot of the transformed data set. To explain 85% of variance in our processed data, we need 1252 principle components. However, our PCA plot will include all 2169 principle components once again, described in Figure 2:

Figure 2: Plot of PCA's using Transformed Data



In Figure 2, we distinctly see the 2169 principle components are largely grouped into 3 separate clusters. Although a PCA plot does not generate clusters, this likely hints of the existence of 3 main brain cell types as described by the scientist. However, as said before, a PCA plot is not a clustering method, and therefore is not guaranteed to reveal clusters. To be more certain that these are 3 separate and distinct clusters, we will fit the PCA plot with labels defined by K-means clustering with 3 as the number of clusters. This will explicitly give us a cluster plot. We get the following plot of Figure 3:

Figure 3: Plot of PCA's with K-Means Clustering with 3 Clusters

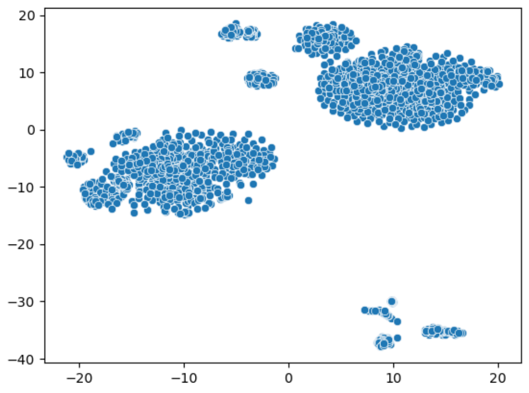


Here, the 3 clusters – each centered around its respective orange dots in the plot, are clearly separated, indicating that all 3 are distinct from each other and that all 3 are distinct brain cell types, as stated before. Each PCA cluster consists of similar samples, which implies that samples from different clusters can vary greatly. In this case, a cell type belonging to a cluster can be very different to another cell from a different cluster – a result consistent with the findings of the scientist.

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.*

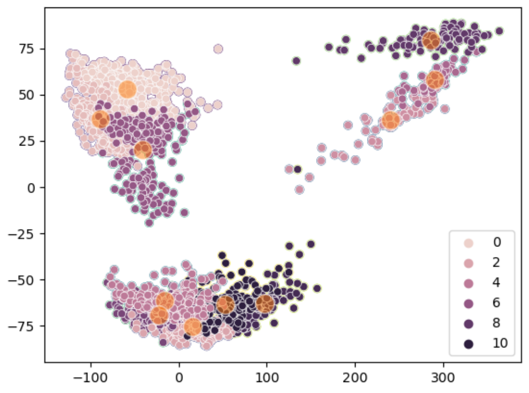
Solution: In the previous problem, we observed that the 3 main clusters form the 3 main cells that can vary greatly from one another, as described by the scientist. In this problem, we will first make a t-SNE plot and then use it for setting the 'number of clusters' parameter in our K-means clustering. Here, we will run multiple t-SNE plots fitted by our PCA plots. We will use the t-SNE plot that generates the lowest observed KL-Divergence value, and this should hint as to what value we should use for K-means clustering. Figure 4 shows the t-SNE plot fitted and transformed with previously calculater PCA plots, that gives the lowest observed value of KL-Divergence at approximately 1.419. We achieved this value by setting number of components to 2 and perplexity to 50 for our t-SNE plots:

Figure 4: t-SNE Plot with the Lowest KL-Divergence



In Figure 4, we see the emergence of 11 clusters located mainly in 3 areas, which indicates the number of clusters we should use for K-means clustering. We fit our PCA plot with K-means set to 11 clusters. Graphing PCA plots and assigning labels identified from K-means clustering, gives the cluster plot shown in Figure 5:

Figure 5: PCA Plot with K-Means Cluster Identities

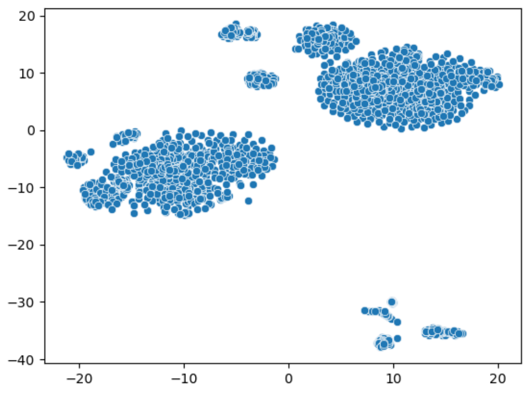


We see that the 3 main clusters are further comprised of smaller clusters – each centered around by its respective orange dots on the plot, which were determined by labels found via K-means. A PCA plot shows clusters of samples based on their similarity. Hence, although these smaller clusters are each different and distinct from one another, they form each of the main clusters, and therefore are grouped by their shared similarity. This indicates that the small subsets of clusters collectively represent their main cluster, which implies that they each belong to the said main cluster. In this case, the different sub-types of a cell, distinct as they may be, overall represent the main cell that they are sub-types of, and belong to it. This result is consistent with the scientist's finding, which describes that a cell may be divided into numerous distinct sub-types, each serving a similar function.

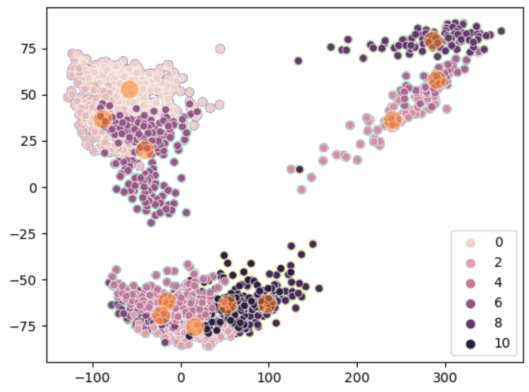
Part 2: Unsupervised Feature Selection

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. (to cluster cells into the subcategories instead of categories)*

Solution: Earlier, we used labels given by K-means clustering with 11 clusters to find the subcategories of each main cell to find Figure 5. The number, 11, was determined via our t-SNE plot with number of components set to 2 and perplexity set to 40. This plot is shown again below.



Observe that the t-SNE plot groups the data into 3 large clusters, and within these clusters, we have smaller clusters. In total, we find 11 clusters in total, which were used to determine the number of clusters in our K-means calculation. Afterwards, the PCA plot of the transformed data were fitted to our K-means. From here, a new plot was made of our PCA plot, by assigning labels found via the calculation of K-means clustering. This gave the final plot found in Figure 5, shown again below:



In Figure 5, we found the 3 main cells, each sub-categorized by 11 clusters labelled by our K-means calculation. Each cluster is centered around its respective orange dots on the plot, and represent separate and distinct sub-types of cells.

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 l_1, l_2, 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.*

Solution: As before, we use the logarithm-transformed data for this problem. We will apply cross-validation splitted 5 times for each of 2 logistic regression models; initially, there was a third model with regularization parameter set to 'elastic net', solver set to 'saga', and l_1 ratio set to 0.5. The solver and l_1 ratio settings were necessary for the selected regularization parameter; however, this model failed to converge before exceeding iteration limits despite setting the highest maximum iteration limit, and therefore was omitted.

The first model will have a regularization parameter l_1; the second model will have a regularization parameter l_2. Both models will have solver set to 'liblinear' and multi_class set to 'ovr'. Afterwards, we will compare the average values of cross-validation scores of each model and choose the model that gives the maximum score. We have the following code:

```
LR1 = LogisticRegression(penalty = 'l1', solver = 'liblinear',
multi_class = 'ovr')
LR2 = LogisticRegression(penalty = 'l2', solver = 'liblinear',
    multi_class = 'ovr')

LR1_cvs = cross_val_score(LR1, X_log2, y, cv = 5)
LR2_cvs = cross_val_score(LR2, X_log2, y, cv = 5)

```

```
LR1_scores = np.sum(LR1_scores) / 5
LR2_scores = np.sum(LR2_scores) / 5
```

Listing 1: Scores of the Models

The average score for the first model, with regularization parameter l_1 and liblinear solver, scored approximately 0.863; the second model scored approximately 0.886. We choose the second model, with regularization parameter l_2 and the liblinear solver, since this method gave a higher score than the other.

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 and 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.*

Solution: We select the features with the top 100 corresponding coefficient values via the following code:

```
log_reg = LR2.fit(X_log2, y)

# sum absolute values of coefficients for each column
values = np.sum(np.abs(log_reg.coef_), axis = 0)

# select indices of the top 100 coefficients
ind = np.argpartition(values, -100)[-100:]
```

Listing 2: Selection of Top 100 Coefficients

From here, we load the evaluation data and perform logarithm transformation to our training and test data. Then, we select data from the evaluation data, corresponding to the top 100 coefficient values found earlier. We train the data to our previously selected model (denoted LR2 in the code) and score the test data:

```
X_train = np.log2(X_train_p2_evaluation + 1)
X_test = np.log2(X_test_p2_evaluation + 1)

LR2.fit(X_train[:, ind], y_train_p2_evaluation)
print(LR2.score(X_test[:, ind], y_test_p2_evaluation))
```

Listing 3: Fitting and Scoring Evaluation Test Data

The score was approximately 0.931. Next, we find the indices of the data corresponding to the top 100 variance and 100 random values in our evaluation training data:

```python
selection_variance = np.var(X_train, axis = 0)

# select indices of the top 100 variance values
var_ind = np.argpartition(selection_variance, -100)[-100:]

# select 100 random numbers from 0 to 45768 and use them as indices
random_values = np.random.randint(45768, size = 100)
```

Listing 4: Finding Top 100 Variance Values and 100 Random Values

Now, we select the data corresponding to the indices of the top 100 variance values and the 100 random numbers we found earlier. We train our model with the data selected and score on the data selected from our test data via the same method:

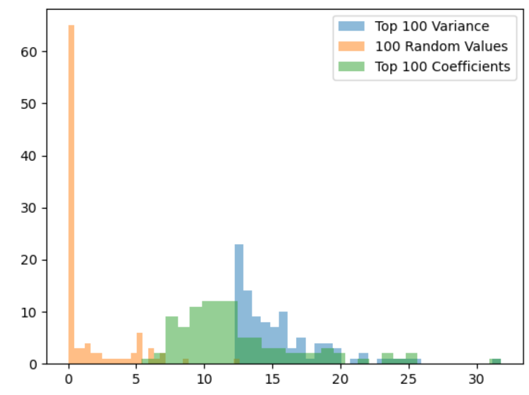
```python
LR2.fit(X_train[:, var_ind], y_train_p2_evaluation)
print(LR2.score(X_test[:, var_ind], y_test_p2_evaluation))

LR2.fit(X_train[:, random_values], y_train_p2_evaluation)
print(LR2.score(X_test[:, random_values], y_test_p2_evaluation))
```

Listing 5: Scoring Top 100 Variance and 100 Random Evaluation Test Data

Our model fitted and tested with data selected from the top 100 variance values scored approximately 0.924. Our model fitted and tested with 100 randomly selected data scored approximately 0.362. The score obtained via using data corresponding to the top 100 coefficients at 0.931 was similar to the score calculated from the data corresponding to the top 100 variance values at 0.924. On the other hand, the score from 100 randomly data at 0.362 was much smaller than the other two.

Now, we compare the variances of the features selected in the above 3 ways; 1) data selected via top 100 coefficients found; 2) data selected via top 100 variances calculated; 3) data selected via 100 random numbers. We get the following histogram:



Here, we observe that histograms of variances of features determined via the top 100 variance calculated and via 100 random values are heavily skewed to the right. On the other hand, the histogram of variances of features selected by the top 100 coefficients are comparatively less skewed.

Problem 3: Influence of Hyper-parameters

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?*

Solution: For this problem, we will use the small data set in the p1 folder, the same data set used for **Problem 1**. First, we apply logarithm transformation to the raw data, and only then transform with PCA. For all of the t-SNE plots in this problem, the number of components will be set to 2 and perplexity set to 40, and those parameters will remain unchanged.

We have the following plots for each of our selection of principle components:

Figure 6: t-SNE Plot with 10 Principle Components, KL-Divergence = 0.255

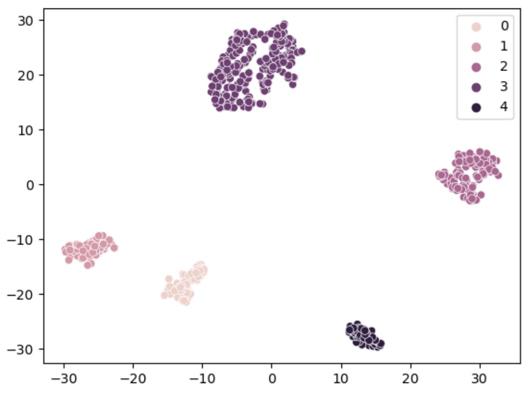


Figure 7: t-SNE Plot with 50 Principle Components, KL-Divergence = 0.382

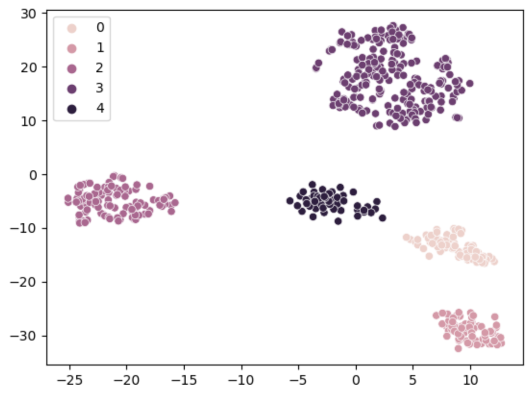


Figure 8: t-SNE Plot with 100 Principle Components, KL-Divergence = 0.513

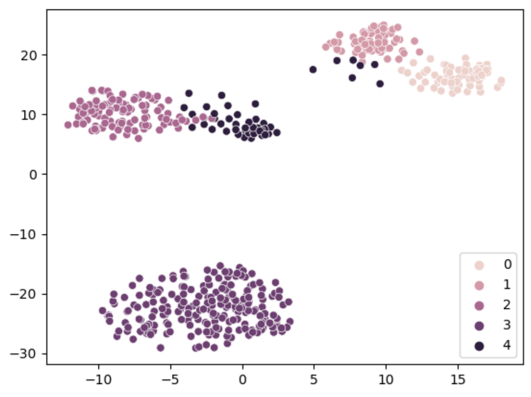


Figure 9: t-SNE Plot with 250 Principle Components, KL-Divergence = 0.685

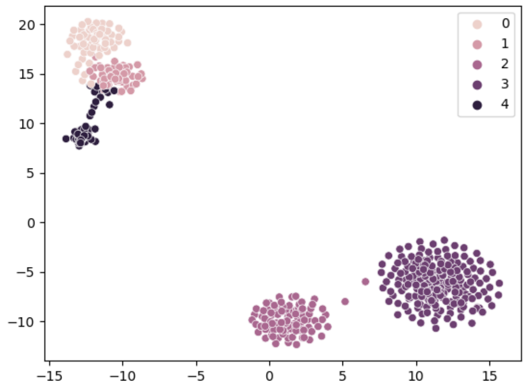
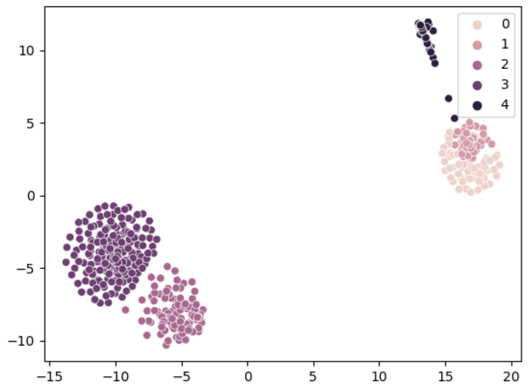


Figure 10: t-SNE Plot with 500 Principle Components, KL-Divergence = 0.988



In the above plots, oberve that as we increase the number of principle components used in our t-SNE plots, KL-Divergence values increase. This indicates that in this case, t-SNE plots become less optimal; this is only true because the number of components and perplexity used for our plots remained consistent and unchanged throughout all of the t-SNE plots conducted here. The only parameter that changed in each plot was the number of principle components, which were fitted and transformed for t-SNE calculation. In general, increasing perplexity decreases KL-Divergence values, since the perplexity parameter is proportional with the variance of the Gaussian used to calculate distances. However, since all other parameters, including perplexity, were kept the same throughout the plots, we observe that from 10 to 500 principle components, increasing the number of principle components lead to less optimal t-SNE plots.

Aside from comparing numerical values of KL-Divergence of each t-SNE plot, we also make a similar observation visually. Initially, in Figure 6, we see compact clusters clearly separated from each other. However, as the number of principle components increase, we see that each cluster slowly disperses from one another. Eventually, some of them begin to combine with each other and become less explicit clusters. This is more evident as we reach higher numbers of principle components in Figures 9 and 10. At this point, most of the clusters have stopped dispersing; however, instead, they are mostly combined with other clusters and become more compressed with each other as the number of principle components increase. This makes it hard to interpret how explicit each cluster is, especially since interpretations of t-SNE plots can be very complicated; in our case, increasing principle components used for t-SNE plots only adds to said complication.

2. (13 points) *Pick three hyper-parameters below (the 3 is the total number that a report needs to analyze. It can take a) 2 from A, 1 from B, or b) 1 from A, 2 from B.) 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. For visualization hyper-parameters, provide substantial visualizations and explanation on how the parameter affects the image. For clustering/feature selection, provide visualizations and/or numerical results which demonstrate how different choices affect the downstream visualizations and feature selection quality. Provide adequate explanations in words for each of these visualizations and numerical results.*

Solution: For this problem, we will work with data sets from the p1 folder, the same data set used for **Problem 1**. We will perform logarithm transformation on the data set, and then fit it to a PCA plot, with all parameters of the PCA plot set to default values. We will use the top 10 principle components calculated from the PCA plot throughout our analysis of t-SNE plots, and the logarithm transformed data for our logistic regression evaluation.

We will examine the effects of the hyper-parameters t-SNE perplexity and learning rate on t-SNE plots, and the effects of L^1 and L^2 regularization parameters on logistic regression; unfortunately, we cannot use elastic net regularization, because it ran into iteration limit problems, despite setting maximum iteration to the highest level. Initially, we will look at these parameters separately; this means that all parameters except the one under observation will be default values, and kept consistent and unchanged. Then, we will conduct experiments jointly between parameters.

First, we consider the t-SNE perplexity parameter by itself. We already know that perpetual increases in perplexity decreases KL-Divergence values, because the perplexity increases with the variance of the Gaussian used to calculate distances. We can easily verify this by making a few plots:

Figure 11: t-SNE Plot with Perplexity = 10, KL-Divergence = 0.477

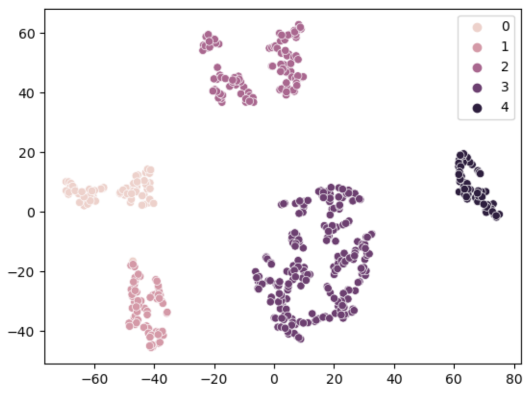


Figure 12: t-SNE Plot with Perplexity = 50, KL-Divergence = 0.224

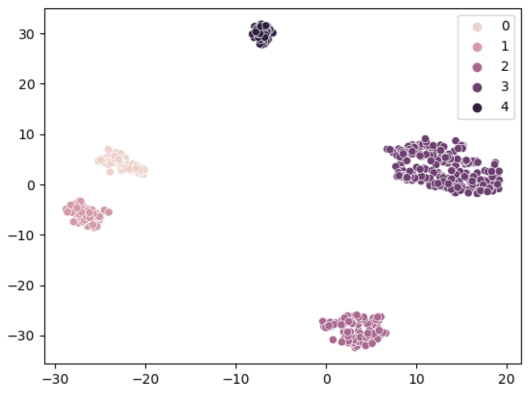


Figure 13: t-SNE Plot with Perplexity = 100, KL-Divergence = 0.119

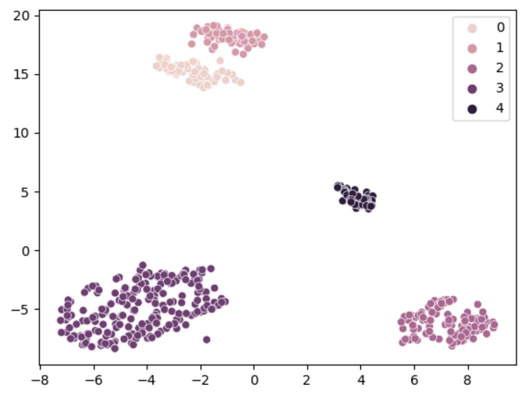
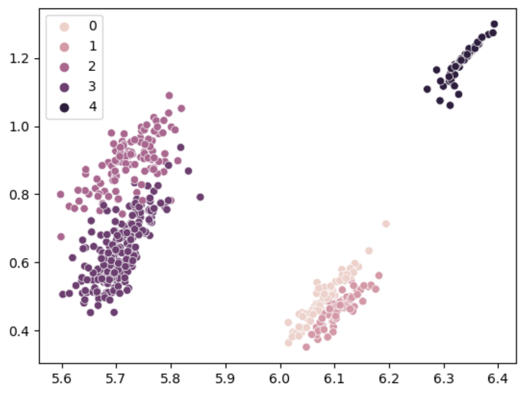


Figure 14: t-SNE Plot with Perplexity = 500, KL-Divergence = 0.002



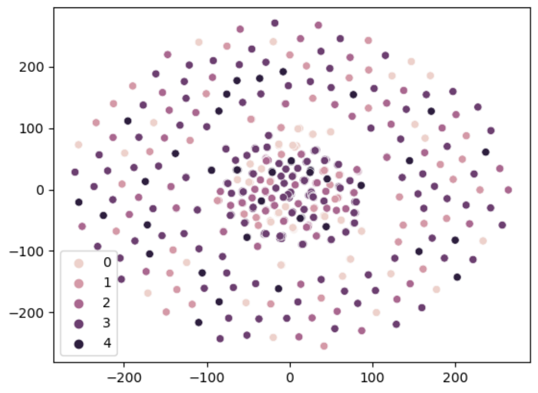
We observe that perplexity is inversely proportional to KL-Divergence; when perplexity is increased, KL-Divergence decreases. However, observe that although increasing perplexity makes better plots initially, t-SNE clusters begin to disperse and combine with each other at high values of perplexity. Notice that neither of the two extremes of perplexity values shown in Figures 11 and 14 make optimal clusters, as they both have highly spread out clusters. Yet, Figure 14 has the lowest KL-Divergence value observed so far. Therefore, using

KL-Divergence values, although ideal for choosing the optimal t-SNE plot of a data set, is not a sufficient indicator to determine the level of perplexity.

Notice also, that although Figures 12 and 13 can be considered as optimal plots, clusters in Figure 13 shows signs of dispersing, which becomes much more pronounced in Figure 14. Figure 12 is the most optimal plot, since all clusters are compactly plotted and each cluster is explicitly distinguished from another.

Consider also, what happens when perplexity becomes extremely low in Figure 15:

Figure 15: t-SNE Plot with Perplexity = 0.1, KL-Divergence = 0.357



We cannot observe any patterns or clusters, as almost every point is spread without any discernible patterns, but the center is heavily clustered. In certain cases, domain knowledge may help find patterns in complicated t-SNE plots. However, an extremely low perplexity value will only make meaningless t-SNE plots. In practice, recommended perplexity values range from 5 to 50 for this purpose; neither an extremely low, nor an abnormally high values of perplexity give optimal t-SNE plots, regardless of KL-Divergence values.

Now, we will examine learning rate of t-SNE plots by itself. We will initially consider learning rate values of 0.1, 100, 1000, 10000. We get the following plots in the next couple pages:

Figure 16: t-SNE Plot with Learning Rate = 0.1, KL-Divergence = 0.563

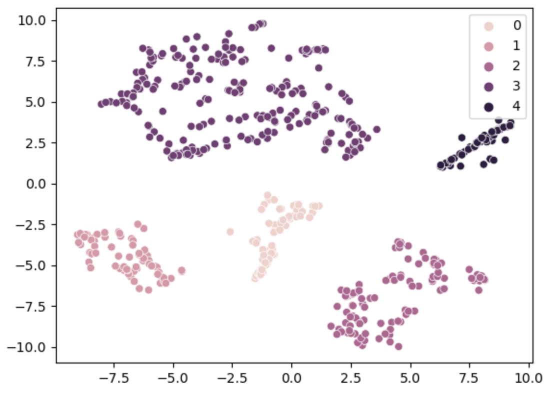


Figure 17: t-SNE Plot with Learning Rate = 100, KL-Divergence = 0.312

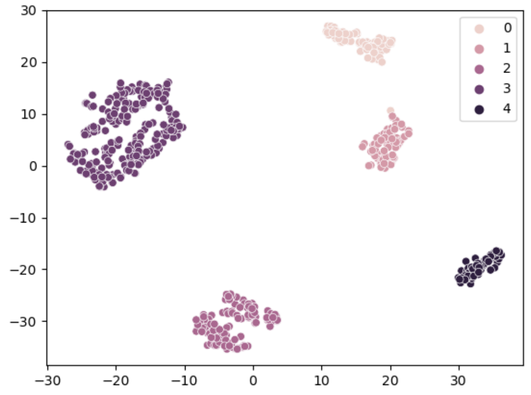


Figure 18: t-SNE Plot with Learning Rate = 1000, KL-Divergence = 0.307

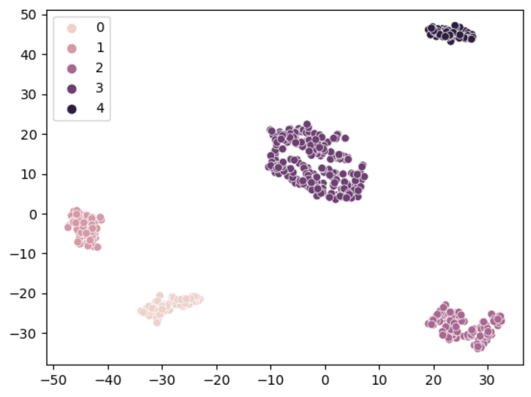
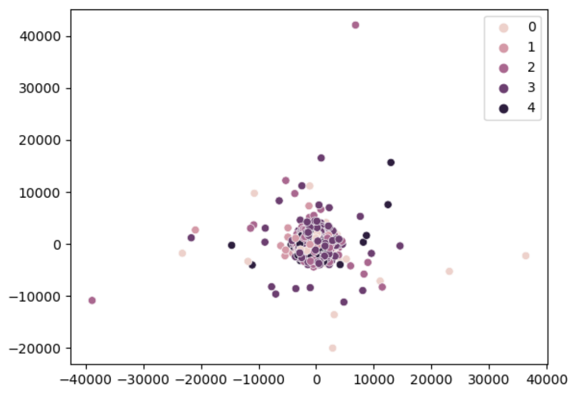
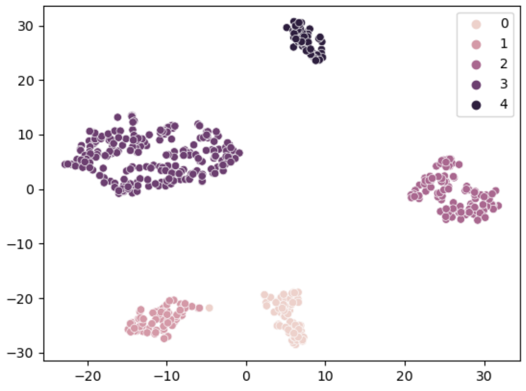


Figure 19: t-SNE Plot with Learning Rate = 10000, KL-Divergence = 3.84



The learning rate controls the size of the increment that gradient descent takes to minimize error. Similar to the perplexity parameter, extreme values on both ends of learning rate lead to undesirable t-SNE plots; for this reason, learning rate generally ranges from 10 to 1000. Observe the values of KL-Divergence within and beyond this range. At learning rate = 100 and 1000, KL-Divergence values converge to approximately 0.3 in Figures 17 and 18. Indeed, if we plot t-SNE with learning rate = 10, we get the following Figure 20:

Figure 20: t-SNE Plot with Learning Rate = 10, KL-Divergence = 0.313



In Figure 20, we have KL-Divergence of 0.313 which is also approximately 0.3. Observe also, that visually, t-SNE plots with learning rates at 10, 100 and 1000 give ideal plots, where different clusters are compact, easily discernible and explicit, as well as being separated from each cluster.

Beyond this range, KL-Divergence values begin to diverge and increase on both ends of the extreme. At learning rate = 0.1 and 10000, we have KL-Divergence values of 0.563 and 3.84 respectively, indicating worse t-SNE plots for extreme values of learning rates.

Aside from the discrepancies in the KL-Divergence values, we can also visually check that t-SNE plots with extreme learning rates result in sub-optimal plots. Notice that at the lowest extreme of learning rate, t-SNE plot becomes incredibly dispersed in Figure 16; at the highest extreme, t-SNE plot condenses into a ball in Figure 19. Consider the following 2 plots of extreme values:

Figure 21: t-SNE Plot with Learning Rate = 0.001, KL-Divergence = 2.75

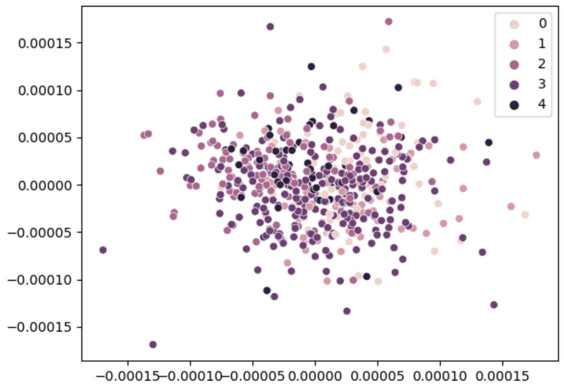
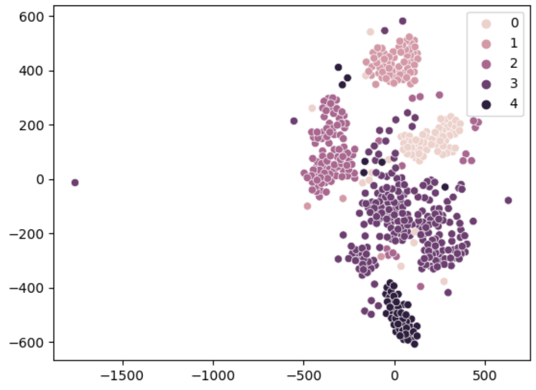


Figure 22: t-SNE Plot with Learning Rate = 5000, KL-Divergence = 1.55



First, consider the KL-Divergence values. In Figure 21, when we take learning rate lower than 0.1 and set it at 0.001, KL-Divergence becomes 2.75 > 0.563 found in Figure 16. In Figure 22 at learning rate = 5000, KL-Divergence is 1.55, which is smaller than 3.84 of Figure 19 at learning rate = 10000. This supports the earlier statement that KL-Divergence values begin to diverge and increase on both ends of the extreme of range 10 to 1000.

Visually, the Figures 21 and 22 do not make ideal t-SNE plots as well. When we take an even lower learning rate value than 0.1 at 0.001 in Figure 21, we see that the clusters, which was dispersing before, are now condensed into a large cloud of indiscernible patterns and sample points. This is likely a result of reaching an unwanted local minimum because of such a small value of learning rate, and in practice, we intuitively increase the learning rate when it is too low.

In Figure 22, samples begin to clump together tightly, and further increases in learning rate, result in a plot much like Figure 19 at learning rate = 10000; we get a very tightly condensed ball of points that is not recognizable as anything else.

In conclusion of the analysis of learning rate, taking values from 10 to 1000 is generally ideal for its selection. In this range, the differences in values for KL-Divergence generally remain negligible; this was true for this experiment, but it may not hold in other scenarios. In this range, we saw a good clustering of explicit samples; anything beyond this range, and we get the Wild West of t-SNE plots. Of course, the ideal value of learning rate changes circumstantially.

Now, we examine the effects of L^1 and L^2 regularization parameters on logistic regression by themselves. The regularization parameters, L^1 and L^2, require specific solvers and therefore, we will use the 'liblinear' solver, since it can handle both types of regularization parameters. All other parameters will be set to default values.

After initializing the different logistic regression models, we will apply cross-validation across 3 splits to each model fitted with the logarithm transformed data sets. Then, we will calculate and compare the average score for each cross-validation. We get the following:

```python
LR1 = LogisticRegression(penalty = 'l1', solver = 'liblinear')
LR2 = LogisticRegression(penalty = 'l2', solver = 'liblinear')

LR1_cvs = cross_val_score(LR1, X_p1_train, y_p1, cv = 5)
LR2_cvs = cross_val_score(LR2, X_p1_train, y_p1, cv = 5)
LR1_ave = np.sum(LR1_cvs) / 5
LR2_ave = np.sum(LR2_cvs) / 5

print(LR1_ave)
print(LR2_ave)
```

Listing 6: Comparing Cross Validation Scores for L^1 and L^2 Regularization

In our case, we get an average score of 1.0 for both regularization parameters. To explore further, we will use a different data set; namely, we will use the one we use for **Probelm 2**, with labels found via K-means earlier. We apply logarithm transformation on the data set as well. We get the following:

```python
LR1 = LogisticRegression(penalty = 'l1', solver = 'liblinear')
LR2 = LogisticRegression(penalty = 'l2', solver = 'liblinear')

LR1_cvs = cross_val_score(LR1, X_log2, y, cv = 5)
LR2_cvs = cross_val_score(LR2, X_log2, y, cv = 5)
LR1_ave = np.sum(LR1_cvs) / 5
LR2_ave = np.sum(LR2_cvs) / 5

print(LR1_ave)
print(LR2_ave)
```

Listing 7: Cross Validation Scores for L^1 and L^2 Regularization for P2 Data Set

For value 'y', we used the labels found via K-means. This is outlined in the code segment below:

```
# fit and transform data via PCA
# apply kmeans with 11 clusters
# t-SNE plot was omitted as it was plotted on another jupyter
# notebook and it takes a long time to plot
pca = PCA().fit(X_log2)
z = pca.transform(X_log2)

X_kmeans = KMeans(11, tol=1e-6)
X_kmeans.fit(z)
y = X_kmeans.labels_
```

Listing 8: Calculation of K-Means using 11 clusters for P2 Data Set

To continue, we get a score of 0.866 for L^1 regularization and a score of 0.891 for L^2 regularization. The fact that we did not see a discrepancy in the p1 data set may be because it is a smaller than the p2 data set, and therefore, did not show noticeable differences. However, in the p2 data set, we see that the L^2 regularization has a slightly higher score than the L^1 regularization. This is likely be because the L^2 regularization reduces weights of the L^2 norm-loss equation:

$$w = min(\sum_{i=1}^{n}[log(1 + e(-zi))] + \lambda * \sum(wj)^2) \tag{1}$$

to values close to 0, whereas the L^1 regularization in the L^1 norm-loss equation:

$$w = argmin(\sum_{i=1}^{n}[log(1 + exp(-zi))] + \lambda * \|w\|) \tag{2}$$

changes those weights to 0. Indeed, we see this difference in the coefficient matrix of the L^1 and L^2 regularization models. Observe the following:

```
log_reg2 = LR2.fit(X_log2, y)
print(log_reg2.coef_)

log_reg1 = LR1.fit(X_log2, y)
print(log_reg1.coef_)
```

Listing 9: Code for Coefficient Matrix of L^1 and L^2 Regularization for P2 Data Set

We get the following outputs:

```
[[-1.35073958e-04  0.00000000e+00 -2.28421150e-03 ...  0.00000000e
    +00
   0.00000000e+00  0.00000000e+00]
[ 8.89635959e-04  0.00000000e+00  3.29745772e-03 ...  0.00000000e
   +00
  0.00000000e+00  0.00000000e+00]
 [-5.04714872e-06  0.00000000e+00 -1.63020865e-03 ...  0.00000000e
   +00
  0.00000000e+00  0.00000000e+00]
 ...
 [-1.41499684e-04  0.00000000e+00  1.24362877e-04 ...  0.00000000e
   +00
  0.00000000e+00  0.00000000e+00]
 [-6.24101165e-06  0.00000000e+00 -1.84929802e-03 ...  0.00000000e
   +00
  0.00000000e+00  0.00000000e+00]
 [-5.65446077e-04  0.00000000e+00  2.64701188e-03 ...  0.00000000e
   +00
  0.00000000e+00  0.00000000e+00]]
```

Listing 10: Coefficient Matrix for L^2 Regularization Model for P2 Data Set

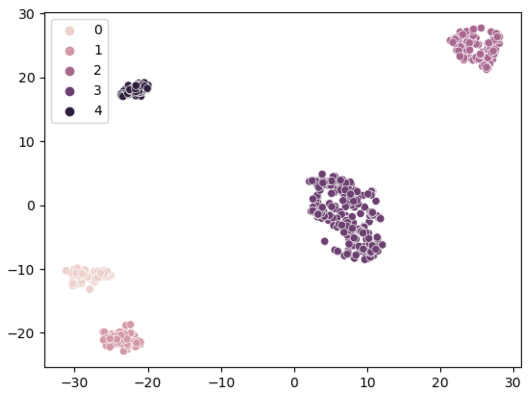
```
[[0.  0.  0.  ...  0.  0.  0.]
 [0.  0.  0.  ...  0.  0.  0.]
 [0.  0.  0.  ...  0.  0.  0.]
 ...
 [0.  0.  0.  ...  0.  0.  0.]
 [0.  0.  0.  ...  0.  0.  0.]
 [0.  0.  0.  ...  0.  0.  0.]]
```

Listing 11: Coefficient Matrix for L^1 Regularization Model for P2 Data Set

Indeed, the matrix generated by L^1 regularization has many zero values, whereas the matrix generated by L^2 regularization has more nonzero values. This does not mean that because L^2 regularization has a higher overall score, it is a better regularization parameter; whether one is better than the other depends on circumstance. In our case, however, L^2 regularization had a stronger impact in our models.

Now, we will return to analyzing hyper-parameters in t-SNE plots. Namely, we will examine what happens when we vary both the perplexity and the learning rate parameters. First, we will use the ideal settings for these parameters found previously. More specifically, our perplexity will be set to 50 and learning rate set to 1000. We will use p1 data sets. We get the following plot in the next page:

Figure 23: t-SNE Plot with Perplexity = 50, Learning Rate = 1000, KL-Divergence = 0.248

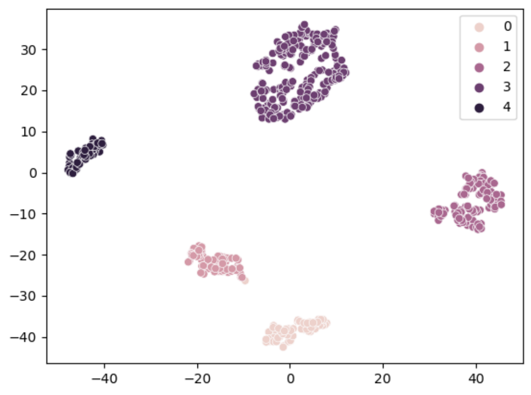


Here, we get a nice t-SNE plot with distinct clusters spaced well apart, and a low KL-Divergence value, which is ideal. In general, finding the ideal values for each parameter before obtaining a final t-SNE plot will lead to optimal results, although it may take considerable time.

We will now see what happens as we vary both parameters. Setting one of the parameters consistent while varying the other will lead to similar results found earlier, when we analyzed the parameters individually. For simplicity, we will make 2 plots where 1) we increase perplexity and lower the learning rate; and 2) we lower the perplexity and increase the learning rate. We will continue to use the p1 data set.

Notice that our parameter values are already respectively set to maximum recommended values. Therefore, we will make a new baseline of comparison of a t-SNE plot with perplexity = 25 and learning rate = 500. We will make comparisons of the plots with this baseline. We have the following:

Figure 24: t-SNE Plot with Perplexity = 25, Learning Rate = 500, KL-Divergence = 0.352



The baseline, Figure 24, also has ideal results. Now, we make the 2 plots mentioned earlier:

Figure 25: t-SNE Plot with Perplexity = 50, Learning Rate = 250, KL-Divergence = 0.226

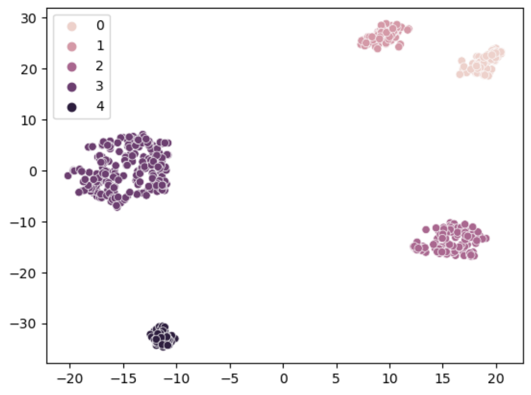
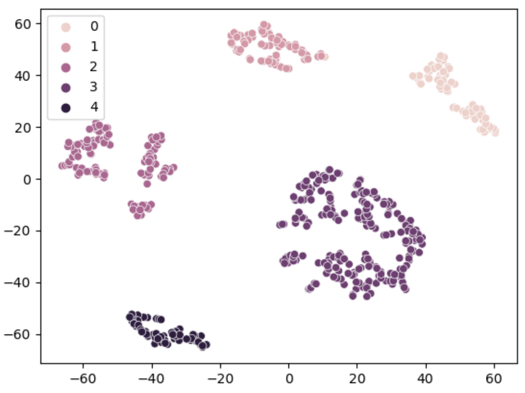
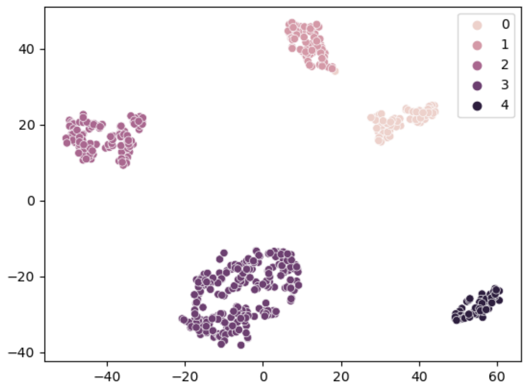


Figure 26: t-SNE Plot with Perplexity = 10, Learning Rate = 750, KL-Divergence = 0.473



We see that both Figures 25 and 26 are generally sufficient plots, although we observe that Figure 26 is starting to disperse. This is likely because of our perplexity value, which is comparatively low. We make this claim intuitively, because 1) perplexity = 10 is very close to the lower bound on the recommended perplexity; 2) when learning rate is too high, samples begin to clump together into a ball, which is not what we observe here. 750 is definitely not too low of a value for the learning rate either, because it is well within the recommended range for learning rates, and also because low learning rate values tend to make t-SNE plots look like a cloud of points, where samples are clustered together all over the plot. This is not what we observe here. To get a better confirmation, we will plot the t-SNE with perplexity slightly increased to 20, but keep the learning rate unchanged at 750. We get the following:

Figure 27: t-SNE Plot with Perplexity = 20, Learning Rate = 750, KL-Divergence = 0.381



As expected, we see that the clusters have become more compact and less dispersed. Furthermore, the KL-Divergence value also decreased from 0.473 to 0.381.

In general, experimenting on data with different selections of individual parameter is ideal for outputting t-SNE plots. For instance, we found multiple ideal plots with low KL-Divergence values throughout this experiment, namely in Figures 12, 17, 18, 20, 23, 24, 25 and 27. Although there is not 1 correct choice of a t-SNE plot, we can find the most realistically ideal plot by specifically choosing different parameters.