Problem Set 07: Unsupervised Learning

Johannes C. B. Dietschreit, Sascha Mausenberger

Due: 25.11.2024

The problems below are meant to be solved using Python functions and libraries. You can do so by either writing short scripts or using Jupyter notebooks. **Report your results in the Quiz on Moodle** to obtain a grade.

Introduction

In this PSet you will work for the first time on unsupervised problems. Unsupervised techniques are important in order to filter out unimportant features and to create more powerful feature-combinations. We will analyze a single-cell RNA-seq dataset, with the goal of unveiling hierarchical structure and discovering important genes. The datasets are all different subsets of a larger single-cell RNA-seq dataset, compiled by the Allen Institute. This data contains cells from the mouse neocortex, a region in the brain which governs higher-level functions such as perception and cognition. The single-cell RNA-seq data comes in the form of a counts matrix, where each row corresponds to a cell and each column to an "experiment" (a different set of cells).

The Problems

1 Dimensionality Reduction and Clustering

1.1 The data sets consist of the count matrix `X.npy` and the cluster labels `y.npy` assigned with a lot of domain knowledge that we do not have. How many clusters have the scientists identified?

(0.5 P)

1.2 They measured 45768 different cells. How many measurements do we have per cell? Do you think that is sufficient?

(0.5 P)

1.3 Let us do the naive thing and train a KMeans model on the data without any pre-processing. However, how many clusters should we choose? As many as the scientists, more, or fewer?

Is the in-built score of the scikit-learn function, which is the negative of the training objective, a good way to determine the best number of clusters? Let's find out.

Train clustering models with $k = 2$ up to $= 50$ (including every integer value in between, this may take a few minutes). For reproducibility use `random_state=42` and `max_iter=500`. Get the score for each of these models and plot it vs. k. Keep in mind that the best score is 0. What number of clusters would give us a score of 0?

(1.0 P)

1.4 Use the kneedle method to determine the "knee" of the within-cluster sum of squares curve (notice that sklearn returns the negative of that sum as score!). Use the provided code. What is k do you get?

```
def rotate_2D(x, theta):
    R = np.array([[np.cos(theta), -np.sin(theta)],
                  [np.sin(theta), np.cos(theta)]])
    return np.einsum("ij,kj->ki", R, x)

def kneedle(wcss, K):
    vecs = np.vstack([K, wcss]).T
    vecs -= vecs[-1]
    vec = np.array([K[0]-K[-1], wcss[0] - wcss[-1]])
    theta = np.arccos(vec[0]/np.linalg.norm(vec))
    new_vec = rotate_2D(vecs, -theta)

    return K[np.argmax(new_vec[:,1])]
```

(0.5 P)

1.5 Train a model with the best number of k as determined in the previous exercise or alternatively with the number used by the scientists. To make your results reproducible, use `random_state=42`, `max_iter=500`, and `init='k-means++'`. In order to understand how well your cluster label assignments correlate with the assignment of scientists compute the normalized mutual information, round your results to 3 digits after the decimal point.

(1.0 P)

1.6 In general, having many more features than data points makes us prone to overfitting. Therefore, we should reduce the number of features before we actually try to do any cluster assignment. We will use principal component analysis to reduce the number of features (retain only those linear combinations of features that contain the most explained variance).

Train a PCA model. Determine how many features need to be kept to retain 95% of the explained variance. `np.cumsum` will be helpful.

(1.0 P)

1.7 Perform k-Means clustering with this significatly reduced number of dimensions with your chosen number of k. Again compute the normalized mutual information to determine the correlation of the cluster assignment with the provided assignments. Has it improved?

(0.5 P)

1.8 We still have less than 10 points per features. We should perform a second round of dimension reduction. This time we will go beyond linear combination of features and use a non-linear method. Use t-SNE and go all the way down to 2 dimensions so that we can easily visualize the data in a plot.

Use `TSNE.fit_transform()` with `perplexity=10` to reduce the number of components to 2. Make a scatter plot of the data. How many clusters do you see?

(1.0 P)

1.9 Use the 2D TSNE data and perform Kmeans clustering with $k = 5$. Make a 2D plot again and color the points by their cluster label. Is the result as you would expect it? Use `random_seed=42` for reproducibility.

(1.0 P)

1.10 Let us try a different form of clustering for comparison. We will use density based clustering, more specifically `DBSCAN`. The parameter `eps` controls the maximal distance that points should be apart for them to be considered in the same cluster. As the points are relatively spread out, we cannot use the default (everything would be considered noise). How many clusters do you get with `eps=5.0`? For your own education, you should also try the non-sensical values of `eps=0.5` and `eps=100.0`.

(1.0 P)

1.11 Using your DBSCAN cluster assignment with `eps=5.0`, let's compare it again to the assignment done by the scientists who had a lot of domain knowledge. Compute the normlaized mutual information again. Has it improved compared to our first clustering attempts?

(0.75 P)

1.12 Plot the t-SNE data colored by the given labels (`y.npy`). How well could you separate all the data points?

(0.75 P)

1.13 To better understand the hyperparameter `perplexity` of t-SNE, we will perform several non-linear dimensionality reductions, using increasing perplexity values. In general, the larger the perplexity the smaller the scale of your points, i.e., they are closer together. The KL-divergence usually decreases with increasing perplexity, meaning the distributions (the original and the t-SNE distirbution) become more similar. As the distance between points changes, one will need very different `eps` values for DBSCAN to cluster the data successfully.

Visually, how many clusters do you expect for perplexity of `500`?

(0.5 P)