1 DBSCAN Clustering

1.1 Theory

DBSCAN stands for "Density-Based Spatial Clustering of Application with Noise". Unlike the k-mean clustering algorithm, DBSCAN can discover clusters of arbitrary shapes, in which each sluter is a maximal set of density-connected points.

DBSCAN works in an iterative manner. It scan randomly points in a given dataset and mark the points as *core points*, *border-points* or *outliers*. If a *core point* is detected, a new cluster with all reachable points from this *core point* is found and saved. This process finishes when all the points are marked.

For a visualization with comments about how DBSCAN works, please see https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

1.2 Python Exercise

In this exercise, you will make a simple implementation of the DBSCAN algorithm and run experiments with it on the digit dataset from sklearn.

Step 1: Load the Digit dataset. Use the function load digits from sklearn to load the dataset. In this exercise, we will only work with samples from 5 classes instead of all the samples in this dataset.

Step 2: Euclidean distance. In this implementation, we will use the euclidean distance as our metric. For your convenience, a function to calculate the distances between pairs of points between two given matrices, in a vectorized form, has been provided.

Step 3: ϵ-neighborhood. In this step, you need to implement a function, named find_eps_neighborhood, to find the ϵ-neighborhood of a given point of interest. The ϵ-neighborhood is defined as all the points which are within a distance ϵ from the point of interest. For convenience in the later steps, you need to return the resulting neighborhood as a set in Python.

Step 4: Reachable points. In this step, a function named `find_reachable_pts` is provided for you. This function call `find_eps_neighborhood` to gradually find all points in the dataset which are reachable from the point of interest, w.r.t. the parameter ϵ.

Step 5: DBSCAN algorithm. After completing some the necessary utility functions in the previous steps, you are all set to write the `DBSCAN` function. You will need to follow the code and fill the missing parts to finish this function.

Step 6: Experiment. In this step, you will run experiments with the `DBSCAN` implementation you have finished in Step 5. First, try with the given parameters for `DBSCAN`, $\epsilon = 20.0$ and minPts $= 10$. You will see a sample visualization of points inside a random cluster, and also the silhouette score for the clustering results.

After that, try yourself with different values of ϵ and minPts. As `DBSCAN` is very sensitive to these two parameters, you shoud see the fluctuation in the results. What are the best values for ϵ and minPts?.

2 Principal Components Analysis

2.1 Theory

Real-world data sets usually exhibit relationships among their variables. These relationships are often linear, or at least approximately so, making them amenable to common analysis techniques. One such technique is PCA, which rotates the original data to new coordinates on which variables are no longer linearly correlated to each other. For a visualization of data before and after using PCA, you can see this article:

`http://setosa.io/ev/principal-component-analysis/`

Given a table of two or more variables, PCA generates a new table with the same number of variables, called the principal components. Each principal component is a linear transformation of the entire original data set. The coefficients of the principal components are calculated so that the first principal component contains the maximum variance (which we may tentatively think of as the "maximum information"). The second principal com-

ponent is calculated to have the second most variance, and, importantly, is uncorrelated (in a linear sense) with the first principal component. Further principal components, if there are any, exhibit decreasing variance and are uncorrelated with all other principal components.

PCA is completely reversible (the original data may be recovered exactly from the principal components), making it a versatile tool, useful for data reduction, noise rejection, visualization and data compression among other things. This exercise walks through the specific mechanics of calculating the principal components of a data set in MATLAB.

Let $\mathbf{X}$ be the $m \times n$ design matrix that contains provides the data set in a structured form. To be specific, m is the number of measurement types and n is the number of samples. We can summarize the PCA algorithm in the following steps:

- Step 1: Remove the mean for each variable, namely subtract the mean for columns.
- Step 2: Compute the covariance matrix $\mathbf{C}_X = \frac{1}{n}\mathbf{X}\mathbf{X}^T$
- Step 3: Find the eigenvectors and eigenvalues of $\mathbf{C}_X$. This could be done using singular value decomposition (SVD) or eigendecomposition. These eigenvectors make up the projection matrix, where a column corresponds to an eigenvector.
- Step 4: Transfrom the original data into new space by multiplying the projection matrix with the original data matrix. Also, you can reduce the dimensionality of the original data in this step by removing a number of eigenvectors before the projection happens.

2.2 Python Exercise

In this exercise, you are going to implement the PCA algorithm. Then, you will transform your data into a new space, where the redundancy in your data is mostly removed. To make it clear, the removal of the redundancy will be visualized.

Step 1: Data. In this step, you will load the toy dataset `digits` from `scikit-learn`, which was used in the previous exercise. Then you have to

center your data by removing the mean of all variables. Be aware that in `numpy`, the data is often organized in rows, namely each row corresponds to an example and each column corresponds to a variable.

Step 2: Covariance matrix. In this step, you will compute the covariance matrix of your data. Again, you should notice that we organize the data differently from the lecture. Therefore, the formula should be modified slightly. Next, you can visualize the redundancy in the covariance matrix using `matplotlib` function `imshow`.

Step 3: Implement PCA. You will implement two functions in this step, namely `pca` and `pca2`. The first function uses SVD while the second one follows eigendecomposition. You are recommended to use functions `svd`[1] and `eig`[2]. Note that for `svd`, you do not need to re-arrange the eigenvectors while you have to do sorting with `eig`.

Step 4: Validation In this step, you will project your original data (mean removed) into a new space using the coefficients you have found in Step 3. Then, the covariance matrix of the projected data will be visualized. At this point, you will see that the redundancy has been removed. You can also reduce the dimensionality of the projected data by removing some eigenvectors before projecting your data.

Step 5: PCA using scikit-learn In this step, you can check if your implementation is comparable to that of `scikit-learn`. You will use the built-in PCA model[3] to transform the original data. Again, the covariance matrix of the transformed data will be visualize to compare with your result from the previous step.

Discussion. PCA "squeezes" as much information (as measured by variance) as possible into the first principal components. In some cases the number of principal components needed to store the vast majority of variance is

[1] https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.svd.html
[2] https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.eig.html
[3] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

shockingly small: a tremendous feat of data manipulation. This transformation can be performed quickly on contemporary hardware and is invertible, permitting any number of useful applications.

For the most part, PCA really is as wonderful as it seems. There are a few caveats, however:

1. PCA doesn't always work well, in terms of compressing the variance. Sometimes variables just aren't related in a way which is easily exploited by PCA. This means that all or nearly all of the principal components will be needed to capture the multivariate variance in the data, making the use of PCA moot.

2. Variance may not be what we want condensed into a few variables. For example, if we are using PCA to reduce data for predictive model construction, then it is not necessarily the case that the first principal components yield a better model than the last principal components (though it often works out more or less that way).

3. PCA is built from components, such as the sample covariance, which are not statistically robust. This means that PCA may be thrown off by outliers and other data pathologies. How seriously this affects the result is specific to the data and application.

4. Though PCA can cram much of the variance in a data set into fewer variables, it still requires all of the variables to generate the principal components of future observations. Note that this is true, regardless of how many principal components are retained for the application. PCA is not a subset selection procedure, and this may have important logistical implications.