MATH 547, PROBLEM SET 2

CALUM MCNAMARA

Problem 1. Assume that we want to determine the elasticity properties of a spring by attaching different weights to it and measuring its length. From Hooke's law we know that the length l depends on the force F according to

$$l = e + kF$$

where e and k are constants to be determined. Assume that we have performed an experiment and obtained the following data

F	1	2	3	4	5
l	7.97	10.2	14.2	16.0	21.2

Calculate the values of e and k by

(a) Minimizing the sum of squares equation.

(b) Using the singular-value decomposition (SVD)

Answers to Problem 1. We first answer part (a), estimating the values of e and k by minimizing the sum of squares equation. Since the textbook Eldén (2019) solves this problem using what might be called the "linear algebra approach," we instead use the standard statistical method as outlined in, e.g., James et al. (2013).[1]

To start, we give a brief overview of our method for minimizing the sum of squares equation. Generally speaking, the goal of minimizing the sum of squares is to predict a variable Y on the basis of a predictor variable X. We assume that there is an approximately linear relationship between X and Y, and note that the equation of this relationship is

$$Y \approx \beta_0 + \beta_1 X. \tag{1}$$

Date: February 20, 2020.

For this problem set, I benefitted most from code-snippets and coding advice from Cooper Stansbury. I also benefited from general discussions with both Cooper Stansbury and Kyle Pettibone.

[1]In this write-up, we minimize the sum of squares equation by hand. However, see the accompanying Notebook for the computation using two different library functions, which validate our results.

Here, β_0 and β_1 are constants, namely the *intercept* and *slope* terms in the linear model. Together, they are known as the *parameters* of the model. Our goal is thus to produce estimates of these parameters, $\hat{\beta}_0$ and $\hat{\beta}_1$, such that our predictions

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \tag{2}$$

are as accurate as possible. (Here, $\hat{y}_i$ is the prediction for Y based on the ith value of X.) We define

$$r_i = y_i - \hat{y}_i \tag{3}$$

to be the ith *residual*, i.e., the difference between the ith observed response value and the ith response value as predicted by our model. We then define the *residual sum of squares* (RSS) as follows:

$$\text{RSS} = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i)^2. \tag{4}$$

Hence, our goal of fitting an accurate model to our data is equivalent to the goal of minimizing the RSS. Some basic calculus (the details of which we omit) shows that the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ which minimize the RSS are

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})r_i}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, \tag{6}$$

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ (i.e., $\bar{y}$ and $\bar{x}$ are the sample means of Y and X, respectively). In other words, these equations define the *least squares coefficient estimates* for our linear model.

Having briefly described our method, we now turn to the data of Problem 1 itself. In this case, we have assumed that the relationship between l and F is linear, and is given by

$$l = e + kF. \tag{7}$$

Thus, the parameters of the model to be estimated are e and k, with e being the intercept term and k being the slope term. Our goal is thus to estimate these terms using the data given in the above table, and according to the method described in the preceding paragraphs. We have five data points to work with, namely

$$(1, 7.97), (2, 10.2), (3, 14.2), (4, 16.0), (5, 21.2).$$

We can plot these points as follows:

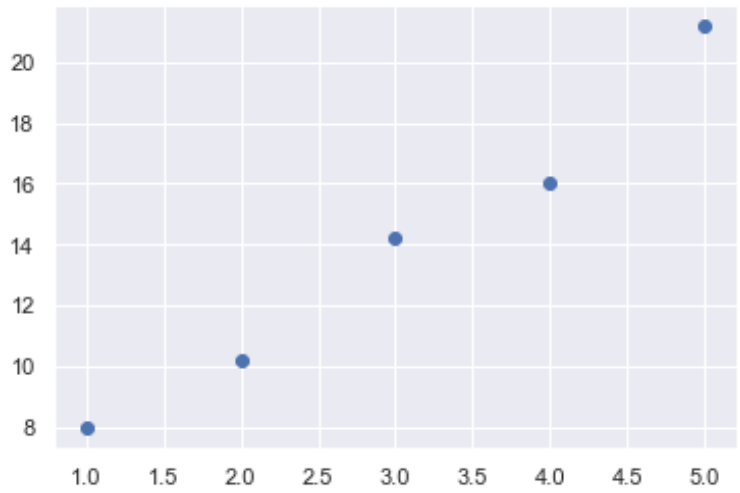

Figure 1. Scatter-plot of the Problem 1 Data.

Let us first find the value of $\hat{k}$, i.e., the "minimizing" estimate of the parameter k. To do so, we find the sample mean of the x-coordinates of our data points, $\bar{F}$. This is

$$\bar{F} = \frac{1}{5}\sum_{i=1}^{5} F_i = 3. \tag{8}$$

Similarly, the sample mean of l, $\bar{l}$, is given by

$$\bar{l} = \frac{1}{5}\sum_{i} y_i = 13.914. \tag{9}$$

Now, some (tedious) computations, which we omit, show that

$$\sum_{i=1}^{n}(F_i - \bar{F})(l_i - \bar{l}) = 32.260. \tag{10}$$

Similarly, we have

$$\sum_{i=1}^{n}(F_i - \bar{F})^2 = 10. \tag{11}$$

Hence, to find the value of $\hat{k}$, we plug these values in as the numerator and denominator of the fraction given in the equation (5), i.e.,

$$\begin{aligned}\hat{k} &= \frac{\sum_{i=1}^{n}(F_i - \bar{F})(l_i - \bar{l})}{\sum_{i=1}^{n}(F_i - \bar{F})^2} \\ &= \frac{32.260}{10} \\ &= 3.226\end{aligned}$$

Having found the value of $\hat{k}$, we can now proceed to find the value of $\hat{e}$. We plug the values of $\bar{l}$, $\hat{k}$, and $\bar{F}$ into equation (6) to give

$$\begin{aligned}\hat{e} &= 13.914 - (3.226 \cdot 3) \\ &= 4.236\end{aligned}$$

Finally, then, the estimates of e and k, $\hat{e}$ and $\hat{k}$, as calculated using the method described above, are 4.236 and 3.226, respectively (which, incidentally, agree with the values given in the textbook). So, the linear model which best fits our data, in the sense of minimizing the RSS, is

$$l = 4.236 + (3.226 \cdot F) \tag{12}$$

We can plot the regression line as follows:

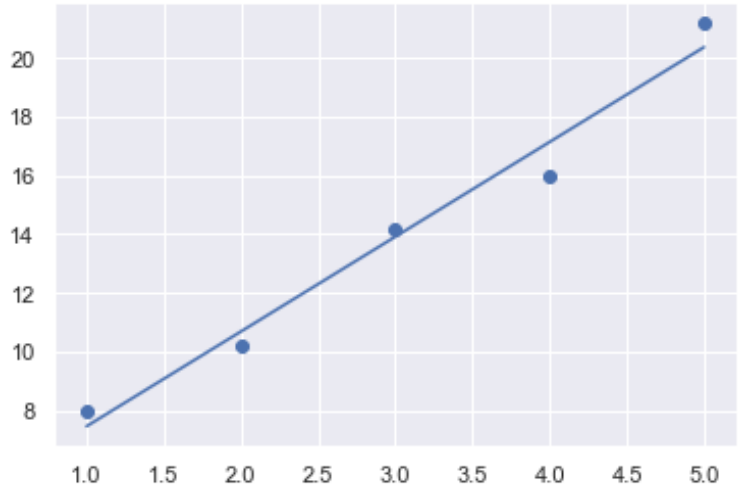

Figure 2. Scatter-plot of the Problem 1 Data with Regression Line.

We now turn to part (b) of this problem, i.e., we calculate the values of e and k using the singular-value decomposition (SVD). To start, we briefly review the singular value decomposition, and we describe how it can be used to find the values of e and k from the problem. Thus, recall that, for any matrix $A \in \mathbb{R}^{m \times n}$, there exist orthogonal matrices $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ and a diagonal matrix $\Sigma \in \mathbb{R}^{m \times n}$ such that

$$A = U \Sigma V^T. \tag{13}$$

The decomposition $A = U\Sigma V^T$ is called the *singular value decomposition* of A. The diagonal entries of Σ, the σ_i, are called the *singular values* of A, and the columns

of U are called the *left singular vectors* of A while the columns of V are called the *right singular vectors* of A.

Now, the SVD can be used to calculate the values of e and k as follows. First, we note that, instead of minimizing the equation (4) from above, we instead seek to minimize the following equation:

$$\min_x ||Fx - l||_2^2. \tag{14}$$

Here, F and l are as before, while x is the vector $[\hat{e}, \hat{k}]^T$ of estimates of e and k. To find x, we note that we need to solve the following equation:

$$x = F^+ l, \tag{15}$$

where F^+ is the *pseudo-inverse* of F. Here, SVD is helpful: it can be used to compute F^+. Hence, we first alter F as follows (noting that this does not change the rank of F):

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$

Then, F^+ can be determined as the product of matrices $V\Sigma^{-1}U^T$. The matrices U and V^T are easily found. However, for clarity we note that Σ^{-1} is the diagonal matrix with entries $\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, ..., \frac{1}{\sigma_n}$.

Having computed the required matrices, we can now solve for x. Just as with our answer to part (a), this computation returns the values $\hat{e} = 4.236$ and $\hat{k} = 3.226$, as expected.[2]

[2]The computations for this problem can be found in the accompanying notebook.

Problem 2. Download the data available on Canvas in the Problem Set 2 folder. This data is described in further detail in Section "MATLAB data description" (below). Perform the 3 spectral clustering algorithms outlined in "A tutorial on spectral clustering" (page 399). Note that this matrix is in the form of a weighted adjacency matrix W.

(a) Plot the results of this clustering (See Figure 1 for an example)

(b) Which algorithm performed the best on this data?

(c) Which algorithm was the fastest? What values for k did you choose and why?

(d) Propose an alternate clustering method other than k-means and plot the results

Answers to Problem 2. The first task we must carry out in answering the questions (a)-(d) of this problem is to run the algorithms described in Von Luxburg (2007) on the `data_mat.csv` data set. As an example, we include the Python code for our first algorithm:[3]

```
def sp_clustering_1():
"""
1. Get adjencency matrix W
2. Compute degree matrix D
3. Compute L (unnormalized)
4. Get first k eigenvectors from L
5. Let k eigenvectors be columns of U
6. TODO
7. Cluster U using k-means

Args:
- k (int): the number of clusters

Returns:
- clustrer labels
"""
w, v = np.linalg.eig(L)
```

[3]This code, as well the code for the t-SNE plotting function and the other two algorithms, is due to Cooper Stansbury. Several library functions are available for these tasks in Python (e.g., through `sklearn` or `pandas`). However, for this problem set, we opted for Cooper's functions instead, as we believe they give a better understanding of what's going on "under the hood."

```
18
19  v = v[:,0:k]
20
21  cluster = KMeans(n_clusters=k).fit(v)
22  return cluster.labels_, v
```

Having run the three algorithms on the given data, we plot the results using a dedicated plotting function, in response to question (a). This function shows the results of the clustering algorithm plotted against of each point, with dimensionality reduced using t-SNE. Recall that t-SNE—or "t-distributed stochastic neighborhood embedding"—works by first constructing a probability distribution over pairs of high-dimensional points such that pairs of "similar" points have a high probability of being chosen while pairs of "dissimilar" points have a low probability of being chosen. Then, t-SNE constructs a second probability distribution over the points in a low-dimensional map, and minimizes the *Kulback-Leibler divergence* between the two distributions. This introduces a certain amount of complication into our visualization. For example, running the Python code may result in different versions of the plots to the ones presented here, since t-SNE is stochastic.[4] In any case, our plots for all three algorithms are as follows:[5]

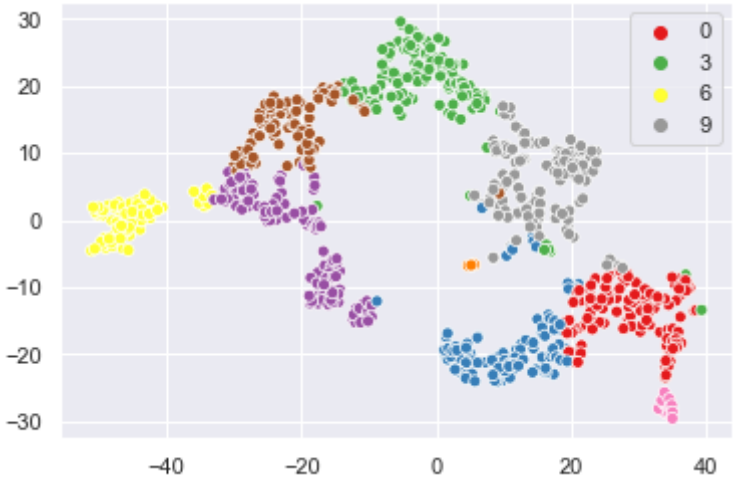

Figure 3. t-SNE Plot for Algorithm 1.

[4]For more information on data visualization using t-SNE, the reader is encouraged to see the paper Maaten and Hinton (2008).

[5]It might be worth pointing out that the plotting function we use here plots the results according to the eigenvector coordinates, rather than according to the original points. This is done for simplicity.

Figure 4. t-SNE Plot for Algorithm 2.

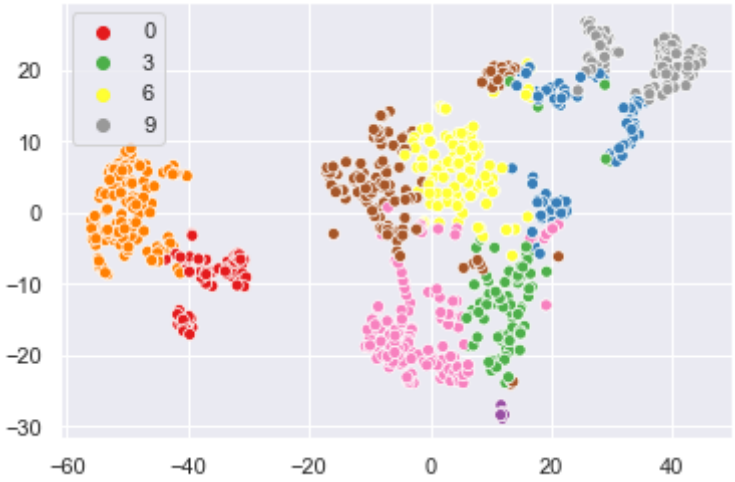

Figure 5. t-SNE Plot for Algorithm 3.

Having plotted our results, we now turn to question (b), i.e., "Which algorithm performed the best on the data?" This is a difficult question to answer, especially since the plotting method we used is stochastic and thus the data visualization varies from iteration to iteration. For example, in the figures above, all three algorithms seem to partition the data into distrinct "neighborhoods." This said, running the visualization algorithm repeatedly does seem to show that algorithms 2 and 3 perform slightly better. Moreover, even in the case of Figures 1-3 above, Figure 2 seems to display slightly less "crossover" than the other two algorithms.

We also experimented with a number of other data visualization methods (though we do not include the results of those experiments in the accompanying notebooks). These other methods did seem to confirm our suspicion that algorithm 1 performed more poorly than either of the other two, and algorithm 2 performed best of all. At this stage, however, a definitive answer as to which algorithm is best is difficult to give.

We now turn to the questions (c), i.e., "Which algorithm performed the fastest?" and "Which values of k did you use and why?" With regards to the first question, we give the results of our speed tests, carried out using Python's "magic function" `timeit`, in the following table:

Algorithm	1	2	3
Time	628 ms	228 ms	1.25 s
Time Deviation	57.9 ms per loop	40.4 ms per loop	95.3 ms per loop

Thus, our tests result in a clear winner and clear loser: algorithm 2 is by far the fastest, while algorithm 3 is by far the slowest. In this latter case, the slowness is likely due to the fact that the algorithm normalizes the data *and* computes the Laplacian matrix. In other words, the algorithm "passes through" the data more times than do the others.

As for why we chose the value $k = 9$ for our algorithms, our answer is based merely on empirical observations. Having experimented with low values, such $k = 3$ and $k = 4$, we found that visually distinct clusters were failing to be "noticed" by the algorithm. Conversely, values of $k > 9$ seemed to suggest that the algorithm was being *too* discerning, i.e., it was partitioning clusters which, visually at least, did not seem to be legitimate. While we would have liked to be able to give a more rigorous answer as to why $k = 9$ is a good value to use, we think that the heuristics described above result in sensible clusterings (though see our answer to (d) for some dissent on this).

Now, the final part of this question asks us to propose an alternative clustering method (other than k-means) to use on the data. Prior to settling on our chosen method, we experimented with a number of clustering algorithms from `sklearn`, such as a function for clustering according to so-called *Gaussian mixture methods*. However, ultimately, we thought an *affinity propogation method* would work best. Code for the algorithm is given below:

```
1 def spectral_affinity():
2 """A function to detect the number of clusters present
```

```
and perform aglorative clustering on results
"""
A = load_data()
_L = scipy.sparse.csgraph.laplacian(A, normed=True)

af = AffinityPropagation(affinity='precomputed').fit(A)
k = int(np.log2(len(set(af.labels_))))
w, v = scipy.linalg.eigh(_L)

_v = normlize_rows(v[:,0:k])
cluster = AgglomerativeClustering(n_clusters=k).fit(_v)

return cluster.labels_, _v
```

Why did we settle on affinity propogation, rather than an alternative method, such as the Gaussian mixture method mentioned above? Our choice was due to the fact that, for each of the above algorithms, we had to prespecify a value for k. Affinity propogation, by contrast, seeks to *detect* an appropriate k-value. A plot of our results using this method is given below. Interestingly, however, using the affinity propogation method resulted in a clustering with only 6 distinct clusterings. This somewhat invalidates the claims made in our answer to (b), i.e., that a value of $k = 9$ suggested itself as the most appropriate.

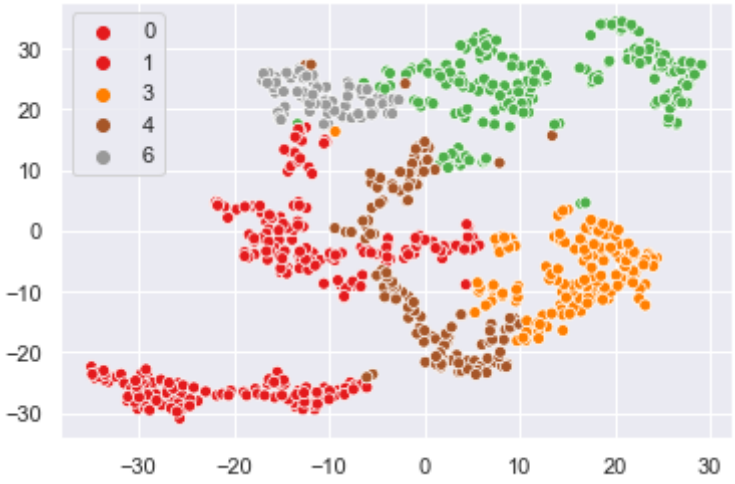

Figure 6. Scatter-plot of the Problem 1 Data.

Concluding Remarks.

Attempting the problems in this problem set was extremely useful in coming to understand different clustering algorithms, as well as the goal of clustering more broadly. We think that (modest) progress has been made towards finding a "best" clustering algorithm for the given data. However, plenty more still needs to be done towards this goal. In particular, we'd like to continue to think about how the most appropriate k-value for the data could be attained, as the "heuristic" method proposed above and the affinity propogation method explored in our answer to (d) seemed to diverge on this question.

References

Eldén, L. (2019). *Matrix methods in data mining and pattern recognition*, volume 15. Siam.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.