Assignment 1

Henrik Hellström
DD2434

December 1, 2020

1 Problem 1

Let $\mathbf{A}$ be a real $n \times n$ symmetrix matrix. We say that $\mathbf{A}$ is positive semidefinite if and only if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$, for every vector $\mathbf{x} \in \mathbf{R}^n$. Prove the following statements, which were claimed in the video lectures.

1.1 (i)

Prove that a real symmetric matrix has real eigenvalues (the point being that a general matrix may have complex eigenvalues). Let $\boldsymbol{\lambda} \in \mathbf{R}^n$ be the eigenvalues of $\mathbf{A}$. Then for any complex-valued vector $\mathbf{u} \in \mathbf{C}^n$, the following must hold:

$$\mathbf{A}\mathbf{u} = \boldsymbol{\lambda}\mathbf{u}. \tag{1}$$

We can take the element-wise complex conjugate of both sides to get a second equality (note that $\mathbf{A}$ is real):

$$\mathbf{A}\mathbf{u}^* = \boldsymbol{\lambda}^*\mathbf{u}^*. \tag{2}$$

Multiply both sides of (1) with $\mathbf{u}^H$ and both sides of (2) with $\mathbf{u}^T$ to get the following system of equations:

$$\begin{cases} \mathbf{u}^H\mathbf{A}\mathbf{u} = \boldsymbol{\lambda}|\mathbf{u}|^2 \\ \mathbf{u}^T\mathbf{A}\mathbf{u}^* = \boldsymbol{\lambda}^*|\mathbf{u}|^2. \end{cases} \tag{3}$$

Since the right side of both equations are scalars, we can take their transpose without any change. Take the transpose of both sides of the first equation in (3):

$$\begin{cases} \mathbf{u}^T\mathbf{A}^T\mathbf{u}^* = \boldsymbol{\lambda}|\mathbf{u}|^2 \\ \mathbf{u}^T\mathbf{A}\mathbf{u}^* = \boldsymbol{\lambda}^*|\mathbf{u}|^2. \end{cases} \tag{4}$$

But since $\mathbf{A}$ is symmetric, the left hand side of both equations in (4) are identical. Therefore:

$$|\mathbf{u}|^2(\lambda - \lambda^*) = 0 \tag{5}$$

and since $\mathbf{u}$ could be any non-zero vector, $\lambda = \lambda^*$ and is therefore real.

1.2 (ii)

Prove that a real symmetric matrix has orthogonal eigenvectors. For any vector $\mathbf{u}_i \in \mathbf{C}^n$ there will be a corresponding eigenvalue $\lambda_i \in \mathbf{R}^1$, such that:

$$\mathbf{A}\mathbf{u}_i = \mathbf{u}_i\lambda_i. \tag{6}$$

Given two eigenvalue-eigenvector pairs, we can use (6) to set up a system of equations:

$$\begin{cases} \mathbf{A}\mathbf{u}_i = \mathbf{u}_i\lambda_i \\ \mathbf{A}\mathbf{u}_j = \mathbf{u}_j\lambda_j. \end{cases} \tag{7}$$

Take the transpose of both sides of the bottom equation in (7):

$$\begin{cases} \mathbf{A}\mathbf{u}_i = \mathbf{u}_i\lambda_i \\ \mathbf{u}_j^T\mathbf{A}^T = \mathbf{u}_j^T\lambda_j. \end{cases} \tag{8}$$

Continue by multiplying the top equation with $\mathbf{u}_j^T$ and the bottom with $\mathbf{u}_i$:

$$\begin{cases} \mathbf{u}_j^T\mathbf{A}\mathbf{u}_i = \mathbf{u}_j^T\mathbf{u}_i\lambda_i \\ \mathbf{u}_j^T\mathbf{A}^T\mathbf{u}_i = \mathbf{u}_j^T\mathbf{u}_i\lambda_j. \end{cases} \tag{9}$$

Since $\mathbf{A}$ is symmetric, the left hand sides of (9) are identical and therefore we are left with:

$$\mathbf{u}_j^T\mathbf{u}_i\lambda_i = \mathbf{u}_j^T\mathbf{u}_i\lambda_j \tag{10}$$

but since $\lambda_i \neq \lambda_j$ and all eigenvectors are non-zero, we conclude that $\mathbf{u}_j^T\mathbf{u}_i = 0$, or equivalently that the eigenvectors are orthogonal.
Then, prove that the eigen-decomposition of a real symmetric matrix $\mathbf{A}$ is $\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^T$. Since the matrix $\mathbf{A}$ is symmetrical, it is diagonalizable. Diagonalizable square matrices always have the following eigen-decomposition:

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} \tag{11}$$

where $\mathbf{Q} \in \mathbf{R}^{n\times n}$ is a matrix with the eigenvectors $\mathbf{u}_i$ as its columns and $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues λ_i as its elements. Since we have already proven that the eigenvectors are orthogonal, we know that $\mathbf{Q}$ is an orthogonal matrix. A property of orthogonal matrices is that $\mathbf{Q}^{-1} = \mathbf{Q}^T$ and therefore

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T. \tag{12}$$

1.3 (iii)

Prove that a positive semidefinite matrix has non-negative eigenvalues. By definition, $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$, for every vector $\mathbf{x} \in \mathbf{R}^n$, if $\mathbf{A}$ is PSD. Since this has to hold for every vector $\mathbf{x}$, it also has to hold for all eigenvectors $\mathbf{u}_i$. Combining the definition of the PSD matrix and the definition of an eigenvector, we get:

$$\mathbf{u}_i^T\mathbf{A}\mathbf{u}_i = \mathbf{u}_i^T\mathbf{u}_i\lambda = |\mathbf{u}_i|^2\lambda \geq 0. \tag{13}$$

Since the euclidean length of any vector is positive, λ has to be non-negative for the inequality to hold.

1.4 (iv)

Let $\mathbf{A}$ be a positive semidefinite matrix. Define matrix $\mathbf{D}$ so that $\mathbf{D}_{ij} = \mathbf{A}_{ii} + \mathbf{A}_{jj} - 2\mathbf{A}_{ij}$. Show that there exists n vectors $\{\mathbf{v}_1, ..., \mathbf{v}_n\}$ in $\mathbf{R}^n$ so that $\mathbf{D}_{ij} = ||\mathbf{v}_i - \mathbf{v}_j||_2^2$ For $\mathbf{D}_{ij} = ||\mathbf{v}_i - \mathbf{v}_j||_2^2$, all elements of $\mathbf{D}$ have to be non-negative. For all diagonal elements it is clearly so because

$$\mathbf{D}_{ii} = \mathbf{A}_{ii} + \mathbf{A}_{ii} - 2\mathbf{A}_{ii} = 0. \tag{14}$$

For the off-diagonal elements ($j \neq i$) we have to prove that this is the case. Since $\mathbf{A}$ is a PSD matrix, its diagonal elements are positive, therefore to show that $\mathbf{D}_{ij} \geq 0$ it is sufficient to show that

$$\mathbf{A}_{ii} + \mathbf{A}_{jj} \geq 2\mathbf{A}_{ij}. \tag{15}$$

For any PSD matrix, every principal sub-matrix is PSD. Therefore

$$|\mathbf{A}_{ij}| \leq \sqrt{\mathbf{A}_{ii}\mathbf{A}_{jj}}. \tag{16}$$

Since the inequality in (15) is trivially true if $\mathbf{A}_{ij}$ is negative, we can assume it is positive for the rest of the proof. With this in mind, we can insert (16) into (15) to get

$$\mathbf{A}_{ii} + \mathbf{A}_{jj} \geq 2\sqrt{\mathbf{A}_{ii}\mathbf{A}_{jj}}. \tag{17}$$

By squaring both sides and simplifying, we get:

$$\mathbf{A}_{ii}^2 + \mathbf{A}_{jj}^2 \geq 2\mathbf{A}_{ii}\mathbf{A}_{jj} \tag{18}$$

which is clearly true for any $\mathbf{A}_{ii}$, $\mathbf{A}_{jj}$, which completes the proof that all elements of the matrix $\mathbf{D}$ are non-negative. Also, since $\mathbf{A}$ is symmetric, $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ and therefore $\mathbf{D}_{ij} = \mathbf{D}_{ji}$. In summary, we know that $\mathbf{D}$ is a hollow matrix (in the sense that the diagonal is zero), it is symmetric, and all its elements are non-zero.
Next, we need to show that n vectors are sufficient to satisfy $\mathbf{D}_{ij} = ||\mathbf{v}_i - \mathbf{v}_j||_2^2$ for all i and j. Since we can choose any vectors $\mathbf{v}_i$, and each vector has n elements, we have n^2 free variables for selecting our $\mathbf{v}_i$. The number of equations to satisfy depends on the number of unique elements in $\mathbf{D}$. In total, $\mathbf{D}$ contains n^2 elements, but the diagonal is zero and the corresponding equations are solved regardless of our choice of $\mathbf{v}_i$, therefore we can remove n equations. For the remaining $n^2 - n$ equations, we can do a further division of 2 since the matrix is symmetric. In total, we will have n^2 free variables and $(n^2 - n)/2$ equations. Since we have more variables than equations there exists a choice $\mathbf{v}_i$ that satisfies all equations.

2 Problem 2

A single SVD operation is not sufficient to perform PCA both on the rows and the columns. A reason for this is that the data centering operation will cause the two data matrices to not simply be the transpose of one another. As an example, let's say we have two data points with three dimensions:

$$\begin{cases} \mathbf{y}_1 = \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} \\ \mathbf{y}_2 = \begin{pmatrix} 7 \\ 1 \\ 1 \end{pmatrix} \\ . \end{cases} \tag{19}$$

After applying the centering operation we get:

$$\begin{cases} \mathbf{y}_1 - \mathbf{E}[\mathbf{y}_1] = \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix} \\ \mathbf{y}_2 - \mathbf{E}[\mathbf{y}_2] = \begin{pmatrix} 4 \\ -2 \\ -2 \end{pmatrix} \\ . \end{cases} \tag{20}$$

The centered data matrix is:

$$\mathbf{Y} = \begin{pmatrix} 0 & 4 \\ -1 & -2 \\ 1 & -2 \end{pmatrix}. \tag{21}$$

Here, the columns are centered but the rows are not. So, it is not true that can just take the SVD of $\mathbf{Y}^T$ to perform PCA on the rows, we would have to redo the centering which would give another matrix.

3 Problem 3

In the derivation of PCA we asked to maximize the expression $\text{tr}(\mathbf{Y}^T\mathbf{W}\mathbf{W}^T\mathbf{Y})$, with respect to matrix $\mathbf{W}$ having k columns, given $\mathbf{Y}$. We claimed that the maximizer is given by $\mathbf{W} = \mathbf{U}_k$, that is, the k left singular vectors of $\mathbf{Y}$ associated with the k largest singular value. Prove this claim.

Using skinny SVD, we can re-express $\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$. The problem can then be expressed as:

$$\begin{aligned} \max_{\mathbf{W}} \quad & \text{tr}(\mathbf{V}^T\boldsymbol{\Sigma}\mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T) \\ \text{s.t.} \quad & \mathbf{W}^T\mathbf{W} = \mathbf{I}. \end{aligned} \tag{22}$$

I added a constraint for the decision matrix to be orthogonal since this is an assumption for PCA (without this constraint, the problem would be unbounded above). As a first step, the cyclic property of the trace operator can be used to get rid of $\mathbf{V}$ and move $\boldsymbol{\Sigma}$. Since $\boldsymbol{\Sigma}$ is a diagonal matrix, I'm using $\boldsymbol{\Sigma}^2 = \boldsymbol{\Sigma}\boldsymbol{\Sigma}$ to denote the element-wise square.

$$\begin{aligned} \max_{\mathbf{W}} \quad & \mathrm{tr}(\mathbf{\Sigma}^2\mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}) \\ \text{s.t.} \quad & \mathbf{W}^T\mathbf{W} = \mathbf{I}. \end{aligned} \tag{23}$$

The trace operator here generates a weighted sum of squares of vector products, where the weights come from the elements in $\mathbf{\Sigma}^2$:

$$\mathrm{tr}(\mathbf{\Sigma}^2\mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}) = \sum_{i=1}^{d}\sigma_i^2\sum_{j=1}^{k}(\mathbf{u}_i^T\mathbf{w}_j)^2 = \sum_{i=1}^{d}\sigma_i^2\sum_{j=1}^{k}|\mathbf{u}_i|^2|\mathbf{w}_j|^2\cos(\theta_{ij}), \tag{24}$$

where σ_i are the diagonal elements of $\mathbf{\Sigma}$, $\mathbf{u}_i$ are the columns of $\mathbf{U}$, $\mathbf{w}_j$ are the columns of our decision matrix $\mathbf{W}$, and θ_{ij} is the angle between vector $\mathbf{u}_i$ and $\mathbf{w}_j$. Because of the constraint ($\mathbf{W}^T\mathbf{W} = \mathbf{I}$), the length of our decision vectors are one ($|\mathbf{w}_j| = 1$), and because of SVD, $\mathbf{U}$ is also an orthogonal matrix and $|\mathbf{u}_i| = 1$.

$$\mathrm{tr}(\mathbf{\Sigma}^2\mathbf{U}\mathbf{W}\mathbf{W}^T\mathbf{U}) = \sum_{i=1}^{d}\sigma_i^2\sum_{j=1}^{k}(\mathbf{u}_i^T\mathbf{w}_j)^2 = \sum_{i=1}^{d}\sigma_i^2\sum_{j=1}^{k}\cos(\theta_{ij}), \tag{25}$$

Our only choice left are the angles θ_{ij} and the best choice is clearly $\theta_{ij} = 0$, which implies colinearity with $\mathbf{U}_k$. Notice though that there are only k vectors in $\mathbf{W}$ but $d > k$ vectors in $\mathbf{U}$ so it's not possible to get a zero angle between all pairs. Therefore, to maximize the expression, the angle should be zero with respect to the vectors $\mathbf{u}_i$ with the largest singular values σ_i. In other words $\mathbf{W} = \mathbf{U}_k$.
Since the remaining $d - k$ vectors are orthogonal to the first k vectors, the remaining vector products are zero. Therefore the objective value of this maximization problem will be:

$$\sum_{i=1}^{d}\sigma_i^2 \tag{26}$$

4 Problem 4

In the derivation of classical MDS with distance matrix, our goal was to derive the Gram matrix (similarity matrix) $\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$ from the distance matrix $\mathbf{D}$, while $\mathbf{Y}$ is unknown.
We get $s_{ij} = -\frac{1}{2}\left(d_{ij}^2 - s_{ii} - s_{jj}\right)$.
We claim that s_{ij} can be computed as $s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$, where d_{1i} and d_{1j} are the distances from the first point in the dataset to points i and j, respectively.
Argue that the claim is correct, that is, it provides the correct estimation for the Gram matrix $\mathbf{S}$.
The claim is that the entries of the similarity matrix can be calculated as follows:

$$\hat{s}_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2). \tag{27}$$

We can re-express this equation as a function of the data vectors $\mathbf{y}_i$, $\mathbf{y}_j$, $\mathbf{y}_0$. If the data vectors are of length d, we get:

$$-2\hat{s}_{ij} = \sum_{a=1}^{d}\left((\mathbf{y}_i)_a - (\mathbf{y}_j)_a\right)^2 + \sum_{a=1}^{d}\left((\mathbf{y}_1)_a - (\mathbf{y}_i)_a\right)^2 + \sum_{a=1}^{d}\left((\mathbf{y}_1)_a - (\mathbf{y}_j)_a\right)^2 \tag{28}$$

where $(\mathbf{y}_i)_a$ is element a of data vector $\mathbf{y}_i$. We can get rid of the a index and the sums by expanding the squares and expressing the resulting expressions as similarities:

$$-2\hat{s}_{ij} = s_{ii} - 2s_{ij} + s_{jj} - 2s_{11} - s_{ii} - s_{jj} + 2s_{1i} + 2s_{1j} \tag{29}$$

which simplifies to:

$$\hat{s}_{ij} = s_{ij} + s_{11} - s_{1i} - s_{1j}. \tag{30}$$

Here I run into a bit of problem, because I'm not sure what is meant when the question states that I should "Argue that the claim is correct" or that it should "provide the correct estimation for the gram matrix". To me, it sounds like I should argue that $\hat{s}_{ij} = s_{ij}$, but I don't believe this to be the case. If so, then the following would have to hold:

$$s_{11} = s_{1i} + s_{1j}, \tag{31}$$

which I believe is impossible to show for a general dataset $\mathbf{Y}$. Maybe I am misinterpreting the question, or I have made a mistake somewhere, but I'm not sure. I cannot proceed.

5 Problem 5

Consider the classical MDS algorithm when $\mathbf{Y}$ is known. In that case, we form $\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$ and obtain the MDS embedding by the eigen-decomposition of $\mathbf{S}$. Show that this procedure is equivalent to performing PCA on $\mathbf{Y}$. In a sense, the singular value decomposition of $\mathbf{Y}$ can be seen as the square root of the eigenvalue-decomposition of $\mathbf{Y}^T\mathbf{Y}$. Consider the SVD of $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{U\Sigma V}^T. \tag{32}$$

We can use this expression to express $\mathbf{Y}^T\mathbf{Y}$:

$$\mathbf{Y}^T\mathbf{Y} = \mathbf{V\Sigma U}^T\mathbf{U\Sigma V}^T = \mathbf{V\Sigma}^2\mathbf{V}^T = \mathbf{V\Lambda V}^T \tag{33}$$

where the final equality comes because the square of the singular values are equal to the eigenvalues. Notice that this expression is exactly equal to the eigen-decomposition of $\mathbf{Y}^T\mathbf{Y}$, so the columns of $\mathbf{V}$ must correspond to the eigenvectors of $\mathbf{Y}^T\mathbf{Y}$.

In PCA, the lower-dimensional data $\mathbf{X}$ is found by $\mathbf{X} = \mathbf{U}_k^T\mathbf{Y}$. If we plug (32) back into that expression we get:

$$\mathbf{X} = \mathbf{U}_k^T\mathbf{Y} = \mathbf{U}_k^T\mathbf{U\Sigma V}^T = \mathbf{1}_{k\times n}\mathbf{\Sigma V}^T. \tag{34}$$

If we instead take the MDS of $\mathbf{Y}^T\mathbf{Y}$ we get:

$$\mathbf{X} = \mathbf{1}_{k\times n}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{V}^T. \tag{35}$$

This is exactly equal to the PCA of $\mathbf{Y}$ because $\mathbf{\Lambda}^{\frac{1}{2}} = \mathbf{\Sigma}$, and $\mathbf{V}$ corresponds to the eigenvectors of $\mathbf{Y}^T\mathbf{Y}$, as established in (33).
In terms of computation, which is the best way to perform the embedding? The computational complexity of approximately finding the eigenvalue-decomposition of an $n \times n$ matrix

is about $\mathcal{O}(n^3)$, however the exact complexity depends on the desired accuracy. The computational complexity of SVD applied to an $m \times n$ matrix is $\mathbf{O}(mn^2)$ if $m \geq n$ (You can take SVD of the transpose otherwise) [1].

In our scenario, we are taking the eigen-decomposition of $\mathbf{Y}^T\mathbf{Y} \in \mathbf{R}^{n \times n}$ and the SVD of $\mathbf{Y} \in \mathbf{R}^{m \times n}$. The SVD will always have the same or lower complexity than the eigenvalue decomposition, therefore PCA is a better way of performing the computation.

6 Problem 6

Argue that the process to obtain the neighbourhood graph G in the Isomap method may yield a disconnected graph.
If the dataset happens to be shaped such that there is a "gap" between two clusters of data-points, it could be that the k closest neighbours are only between other points within the same cluster. As an example, I've drawn an ugly dataset in paint where this is the case, see Figure 1. Here, there is a gap between one cluster at the top-left of the figure and the rest of the data set. To illustrate the disconnection, I've drawn the graph formation at two points on the border if $k = 3$. For all points in this dataset, the three closest points belong to its own cluster, therefore there will be no connection between the two clusters.

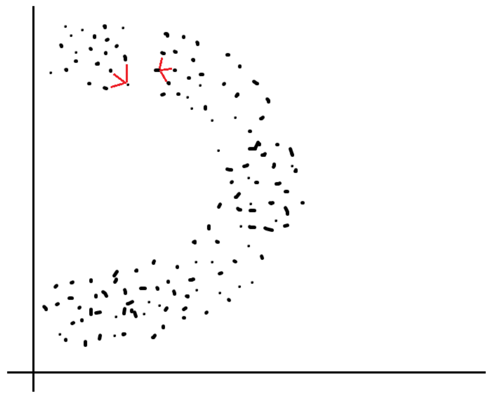

Figure 1: Isomap dataset example. Black dots = data points, red lines = three nearest neighbours. With $k = 3$ and this dataset, the isomap algorithm would fail since the resulting graph would be disconnected.

Propose a heuristic to patch this problem. Justify your heuristic. The reason that the two areas are not connected, is because for the chosen k, there are k points within each points own cluster that are closer than the other cluster. However, if k is big enough, there will always be a connection. Even if there is an enormous distance between the two clusters, if k is equal to the number of points in the smaller cluster, there will always be a connection, see Figure 2.
Since a sufficiently large k will always lead to a connected graph, we can increase k by one ($k \longleftarrow k + 1$) if the path algorithm fails. We then have to recreate the entire graph, and try

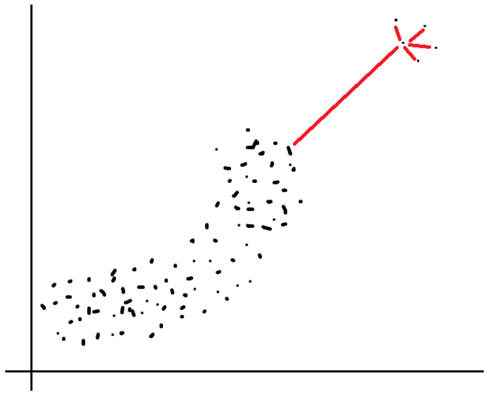

Figure 2: Isomap dataset example. Black dots = data points, red lines = five nearest neighbours. With $k = 5$ the very distant cluster gets connected, because there are only 5 points in the smallest cluster.

the path finding algorithm again. Eventually, we will have a connected graph. The heuristic has two potential problems: 1) it can be computationally expensive since the entire graph has to be reconstructed multiple times, and 2) for certain datasets, k could get very large, which would create unwanted scenarios where the euclidean distance approaches the geodesic distance, even for distant points. Even so, it is a nice simple heuristic that will always yield a connected graph, and should work quite well on some datasets. In the dataset of Figure 1, the heuristic would probably yield a connected graph at $k = 8$ or so, which seems acceptable.

7 Problem 7

7.1 Code

The code for this problem can be found at: Github hyperlink

7.2 Preprocessing

Since PCA is originally designed for real values and I didn't want to make any major changes, I decided to treat all values as reals, where the booleans simply have a value of 0 or 1. For the "legs" attribute, I normalized it to be between 0 and 1 by simply dividing its value by 8.

$$\text{legs} \longleftarrow \frac{\text{legs}}{8}. \tag{36}$$

I created a matrix $Y \in \mathbf{R}^{16\times 101}$ using all attributes except the "name" and the "type". The dimensions follow the standard set in the lectures, where each column corresponds to one data point. Next, I centered the matrix:

$$\mathbf{Y} \longleftarrow \mathbf{Y} - \frac{1}{101}\mathbf{Y}\mathbf{1}_{101}^T\mathbf{1}_{101}. \tag{37}$$

7.3 PCA

For PCA, I made my own implementation, but I took an SVD implementation from numpy, so it was extremely simple. Basically, after preprocessing I just use numpy's SVD implementation and the take the first 2 columns of $\mathbf{U}$ to form $\mathbf{U}_k$ (numpy already orders columns by singular value size). I form the 2-dimensional data by taking:

$$\mathbf{X} = \mathbf{U}_k^T Y. \tag{38}$$

Since $\mathbf{X}$ is two-dimensional it can easily be plotted, see Figure 3. The algorithm actually does very well, there are clear clusters for some of the species, especially mammals are seperable by a linear separator from the other species. Also, fish and amphibians form clear distinctions from the rest.

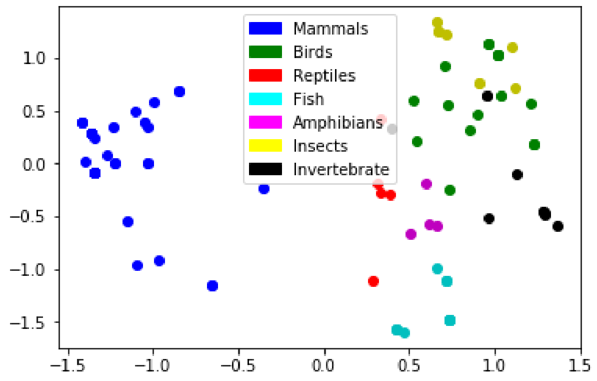

Figure 3: Plot of X created using PCA. The 7 types have been named according to my best guess by looking at what I believe to be the species of the animals in each type (I guess vampires are mammals?).

To gain some insight, I tried to look at the largest element of U_k (in terms of absolute value) since this element would have a big impact on the positioning of each animal. In my case it turned out to be "milk" which negatively affected the first component by 0.46. This can clearly be seen in Figure 3, where the mammals are much further to the left, and they should also be the only animals with the milk variable set to 1.
Finally, I also annotated the points with the name of each species, seen in Figure 4. One interesting thing to note here is that "platypus" is the mammal furthest away from its cluster, this makes sense since the platypus is famous for being a mammal with very "un-mammal" attributes (such as being an egg-laying mammal). Also, we see that dolphin, seal, and sealion form a sub-cluster within the mammals, which is reasonable since they are the only aquatic mammals in the dataset. Also, as can be predicted from Table 1, they are further south (lower Y value).

7.4 MDS

To infer the importance of different attributes, we can take another look at the $\mathbf{U}_k$ matrix created in the previous subsection, summarized in Table 1. As discussed previously, milk had

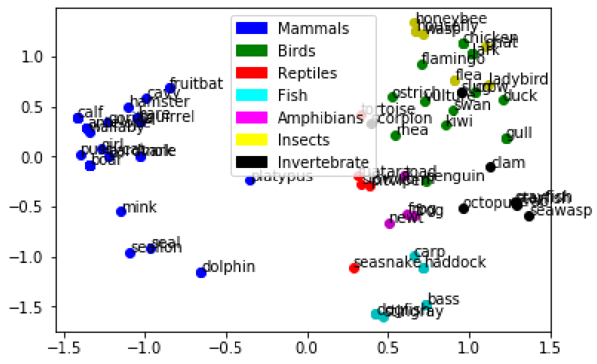

Figure 4: Same plot as the previous figure but with annotated points.

Attribute	X	Y
Milk	-0.46	0.03
Hair	-0.43	0.15
Aquatic	0.19	-0.47
Predator	0.02	-0.38

Table 1: Table of attributes with the biggest values in $\mathbf{U}_k$, milk and hair are the biggest in the x-axis, aquatic and predator are the biggest in the y-axis.

the biggest impact on the x-axis, and we could clearly see in the plot that mammals (which produce milk) were further to the left. To do MDS, I start by calculating the pairwise distance of all animals and form the distance matrix $D \in \mathbf{R}^{101\times101}$. As an example of distances, I took found two pairs that intuitively should be similar and different:

$$\text{difference}(\text{seal}, \text{sealion}) = 1.06 \tag{39}$$

$$\text{difference}(\text{dolphin}, \text{honeybee}) = 13.6, \tag{40}$$

we can see that the intuition was correct in this scenario, the full distance matrix $\mathbf{D}$ is too large to type out here.
To run the MDS algorithm, I followed the instructions of the course book [2]. First, I form the S matrix via double centering:

$$\mathbf{S} = -\frac{1}{2}\left(\mathbf{D} - \frac{1}{n}\mathbf{D}\mathbf{1}_n\mathbf{1}_n^T - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T\mathbf{D} + \frac{1}{n^2}\mathbf{1}_n\mathbf{1}_n^T\mathbf{D}\mathbf{1}_n\mathbf{1}_n^T\right). \tag{41}$$

Then, I find the eigen-decomposition of $\mathbf{S}$ as:

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T. \tag{42}$$

Finally, the 2-dimensional data is formed using the eigen-decomposition and a truncated eye matrix:

$$\hat{\mathbf{X}} = \mathbf{I}_{2\times n}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^T. \tag{43}$$

I had to make a slight modification here because there would be some eigenvalues that were so small that they caused the square-root to crash. In reality, these are not actual

eigenvalues, but since the eigen-decomposition algorithm is approximate, it finds extremely small eigenvalues, so I added a truncation of values below 10^{-3} and set them all to zero. Then I got 16 eigenvalues, as expected. I plotted the results in Figure 5.
The results are identical to PCA, except for the mirroring around the x-axis. It is expected that they should give the same result, both are linear mappings that minimize the expected error, and we proved that they are identical in an earlier problem. The advantage of using MDS over PCA is that you don't need to know the dataset $\mathbf{Y}$, but it's enough to know either the similarities or distances. But since we have full information in this case the results are the same. Also, since we have the full dataset, using PCA is more computationally efficient.

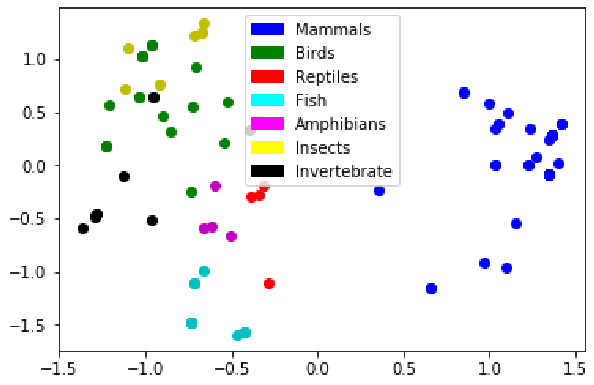

Figure 5: Plot of X created using MDS. The 7 types have been named according to my best guess by looking at what I believe to be the species of the animals in each type (I guess vampires are mammals?).

7.5 Isomap

For the isomap algorithm, we now have a design parameter p which determines the number of nearest neighbours used to form the graph. My implementation of isomap exactly follows the description from slide 68 on module 2 from the course. Which is the following:

1. Compute a distance matrix $\mathbf{D}$ using the differences in the original space $\delta_{ij} = ||\mathbf{y}_i - \mathbf{y}_j||^2$
2. Construct a graph $\mathbf{G}$, where vertices represent points, and each point is connected to its p nearest points, according to the distance matrix $\mathbf{D}$. (To represent the graph $\mathbf{G}$ I created an n by n matrix where the distance of connected points were taken directly from $\mathbf{D}$ and unconnected points were given a high value to symbolize infinity)
3. For each pair of points i and j compute the shortest path distance d_{ij} using i to j on the graph G. (I used the Floyd-Marshall algorithm)
4. Use MDS on the shortest-path distances $\{d_{ij}\}$ to compute the low-dimensional embedding $\mathbf{X}$.

To find a suitable design parameter, I simply experimented with a few different values. For $p < 17$, the algorithm failed since the graph was not connected, so results for that is not included. For exactly $p = 17$, the results are presented in Figure 6. There are some interesting differences here compared to our linear models, first of all we notice that the platypus has moved even further away from the mammals and have approached the birds. Also, other species are forming clearer clusters, for instance the birds are now close together but were interspersed with insects in the linear model. It's hard to say which is better since there is no clear metric to go by here, but in my own opinion I think the isomap does a better job of separating the different species and thereby visualizing the data.

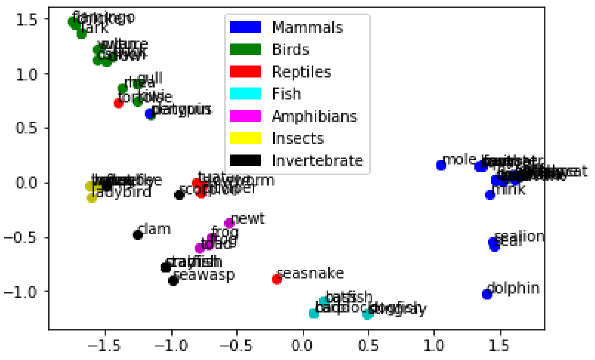

Figure 6: Plot of X created using isomap with $p = 17$. The 7 types have been named according to my best guess by looking at what I believe to be the species of the animals in each type (I guess vampires are mammals?).

If the p parameter is increased, the model should approach the linear case, in fact, if $p = n = 101$, we should exactly recreate MDS, because all points can be reached within one hop. As a sanity checked I tried this out, and it does give the same result (although mirrored against the x-axis). To get an idea of the middle ground, I ran the script once for $p = 30$, plotted in Figure 7. Here, the platypus is once again closer to the mammal cluster while most other species are still separated. Just like in the previous figure, the reptiles seem to be the hardest species to cluster together. In my opinion, this final figure seems to be the best visualization of the data.

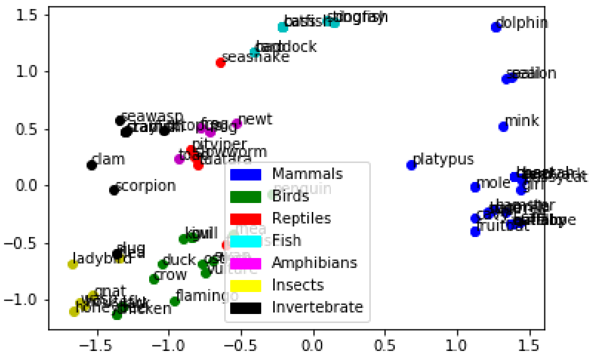

Figure 7: Plot of X created using isomap with $p = 30$.

References

[1] V. Y. Pan and Z. Q. Chen, “The complexity of the matrix eigenproblem,” in *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, 1999, pp. 507–516.

[2] J. A. Lee and M. Verleysen, *Nonlinear dimensionality reduction.* Springer Science & Business Media, 2007.