Assignment 1 - DD2434 Machine Learning, Advanced Course

August Regnell 970712-9491

18 November, 2020

Contents

1 Problem 1

1.1 Problem 1.1

Prove that a real symmetric matrix has real eigenvalues.

Proof: Suppose $\mathbf{A}$ is a real symmetric matrix and $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{A}$. Thus $\mathbf{A} = \mathbf{A^T} = \mathbf{A}^*$ where $*$ denotes the complex conjugate transpose of the matrix. Then there exists a nozero vector $\mathbf{x} \in \mathbb{C}^n$ such that $\mathbf{Ax} = \lambda\mathbf{x}$. It follows from taking the complex conjugate of both sides that $\mathbf{A}\bar{\mathbf{x}} = \bar{\lambda}\bar{\mathbf{x}}$. Therefore,

$$\lambda\mathbf{x^T}\bar{\mathbf{x}} = (\mathbf{Ax})^\mathbf{T}\bar{\mathbf{x}} = \mathbf{x^T}\mathbf{A}\bar{\mathbf{x}} = \mathbf{x^T}(\bar{\lambda}\bar{\mathbf{x}}) = \bar{\lambda}\mathbf{x^T}\bar{\mathbf{x}}$$

which implies that $\lambda = \bar{\lambda}$, so $\lambda \in \mathbb{R}$.

■

1.2 Problem 1.2

Prove that a real symmetric matrix has orthogonal eigenvectors. Then, prove that the eigen-decomposition of a real symmetric matrix $\mathbf{A}$ is $\mathbf{A} = \mathbf{U\Lambda U^T}$.

Proof: Let $(\lambda_i, \mathbf{u}_i)_{i=1,\dots,n}$ be the pairs of eigenvalues and eigenvactors of the symmetrical $n \times n$ matrix $\mathbf{A}$. We then get the eigenvector equation

$$\mathbf{Au_i} = \lambda_i \mathbf{u_i} \tag{1}$$

Left multiplying by $\mathbf{u}_j^T$ gives

$$\mathbf{u_j^T A u_i} = \lambda_i \mathbf{u_j^T u_i}$$

and hence, by exchanging the indices, we have

$$\mathbf{u_i^T A u_j} = \lambda_j \mathbf{u_i^T u_j}$$

We know take the transpose of the second equation and use $\mathbf{A}$'s symmetrical property $\mathbf{A^T} = \mathbf{A}$, and subtract the two equations to get

$$(\lambda_i - \lambda_j)\mathbf{u_i^T u_j} = 0$$

Hence, for $\lambda_i \neq \lambda_j$, we have $\mathbf{u_i^T u_j} = 0$, and hence $\mathbf{u_i}$ and $\mathbf{u_j}$ are orthogonal. If the two eigenvalues are equal, then any linear combination $\alpha\mathbf{u}_i + \beta\mathbf{u}_j$ is also an eigenvector with the same eigenvalue, so we can select one linear combination arbitrarily, and then choose the second to be orthogonal to the first. Hence the eigenvectors can be chosen to be orthgonal, and by normalizing can be set to unit length.

We can take the eigentvectors $\mathbf{u}_i$ to be the columns of an $n \times n$ matrix $\mathbf{U}$, which from orthonormality satisfies

$$\mathbf{U^T U} = \mathbf{I}$$

This matrix is orthogonal, which also implies that its rows are orthogonal, $\mathbf{UU^T} = \mathbf{I}$. This further implies, using the previous equation, that $\mathbf{U^T U U^{-1}} = \mathbf{U}^{-1} = \mathbf{U^T}$, meaning that the transpose of $\mathbf{U}$ is its inverse.

The eigenvector equation, (1), can be expressed in terms of $\mathbf{U}$ on the form

$$\mathbf{AU} = \mathbf{U\Lambda}$$

where $\mathbf{\Lambda}$ is an $n \times n$ matrix whose diagonal elements are given by the eigenvalues λ_i. Right multiplying both sides with $\mathbf{U^T}$ gives

$$\mathbf{U\Lambda U^T} = \mathbf{AUU^T} = \mathbf{AUU^{-1}} = \mathbf{A}$$

■

1.3 Problem 1.3

Prove that a positive semidefinite matrix has non-negative eigenvalues.

Proof: Let $(\lambda_i, \mathbf{u}_i)$ be pairs of eigenvalues and eigenvactors of the positive semidefinite matrix $\mathbf{A}$. Then by defintion we have

$$\mathbf{A}\mathbf{u_i} = \lambda_i \mathbf{u_i}$$

and

$$\mathbf{x^T A x} \geq 0$$

for all $\mathbf{x}$.

If $\mathbf{u_i}$ is an eigenvector of $\mathbf{A}$, then

$$\mathbf{u_i^T A u_i} = \mathbf{u_i^T}(\lambda_i \mathbf{u_i}) = \mathbf{u_i^T u_i}\lambda_\mathbf{i}$$

Since $\mathbf{u_i^T u_i}$ is a positive number, λ_i must be greater than or equal to 0 for $\mathbf{u_i^T A u_i} \geq 0$ to hold, meaning that λ_i is non-negative for all i.

■

1.4 Problem 1.4

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 exist n vectors $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ in $\mathbb{R}^n$ so that $\mathbf{D}_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|_2^2$.

Proof: As was shown in a Problem 1.2 the eigen-decomposition of a real symmetric matrix $\mathbf{A}$ is $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathbf{T}$. Since $\mathbf{\Lambda}$ is a diagonal matrix with positive diagonal elements one can define its square root $\mathbf{\Lambda}^{\frac{1}{2}}$ with $\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{\Lambda}^{\frac{1}{2}} = \mathbf{\Lambda}$. We can then define $\mathbf{V}$ as

$$\mathbf{V} = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}$$

which gives

$$\mathbf{V}\mathbf{V}^\mathbf{T} = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{U}^\mathbf{T} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\mathbf{T} = \mathbf{A}$$

Choosing $\mathbf{v}_1, \ldots, \mathbf{v}_n$ to be the rows of $\mathbf{V}$ means that $\mathbf{A}_{ij} = (\mathbf{V}\mathbf{V}^\mathbf{T})_{ij} = \mathbf{v}_i^T\mathbf{v}_j$. Expanding $\|\mathbf{v}_i - \mathbf{v}_j\|_2^2$ gives

$$\mathbf{D}_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\|_2^2 = (\mathbf{v}_i - \mathbf{v}_j)^T(\mathbf{v}_i - \mathbf{v}_j) = \mathbf{v}_i^T\mathbf{v}_i + \mathbf{v}_j^T\mathbf{v}_j - 2\mathbf{v}_i^T\mathbf{v}_j =$$

$$= (\mathbf{V}\mathbf{V}^\mathbf{T})_{ii} + (\mathbf{V}\mathbf{V}^\mathbf{T})_{jj} - 2(\mathbf{V}\mathbf{V}^\mathbf{T})_{ij} = \mathbf{A}_{ii} + \mathbf{A}_{jj} - 2\mathbf{A}_{ij}$$

■

2 Problem 2

Consider a data matrix of dimension $m \times n$. In some applications the role of points and dimensions can be interchanged. For example, given a document corpus represented as a matrix of type "documents $\times$ words", we may want to analyze documents based on which words occur in them, or we may want to analyze words based on which documents they appear in. So it is meaningful to perform PCA both with respect to the rows on a matrix gives also the SVD on its transpose.

As we discussed in the video lectures, PCA relies on SVD. Moreover, since $(\mathbf{U\Sigma V}^T)^T = \mathbf{V\Sigma}^T\mathbf{U}^T = \mathbf{V\Sigma}'\mathbf{U}^T$, where Σ' differs from Σ only in terms of size, performing SVD on a matrix gives also the SVD on its transpose.

Does this argument imply that a single SVD operation is sufficient to perform PCA both on the rows and the columns of a data matrix?

Justify your answer.

2.1 Solution

Performing PCA on $\mathbf{Y}$ is equivalent to performing singular value decomposition on $\mathbf{Y}$, taking $\mathbf{U}_k$ to be the k left singular vectors corresponding to the k largest singular values and mapping the data to k-dimensional space by $\mathbf{X} = \mathbf{U}_k^T\mathbf{Y}$. Since $\mathbf{Y}^T = \mathbf{V\Sigma}^T\mathbf{U}^T$, the left singular vectors of $\mathbf{Y}^T$ is the right singular vectors of $\mathbf{Y}$, meaning that the mapping to k-dimensional space for $\mathbf{Y}^T$ is done as follows: $\mathbf{X} = \mathbf{V}_k^T\mathbf{Y}^T$. Since $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{\Sigma}$ (and $\mathbf{\Sigma}'$) are calculated during the singular value decomposition for the PCA of both $\mathbf{Y}$ and $\mathbf{Y}^T$, a single SVD is sufficient to perform PCA bothon the rows and the columns of a data matrix.

■

3 Problem 3

In the derivation of PCA we asked to maximize the expression $tr(\mathbf{Y^T W W^T Y})$, with respect to a matrix $\mathbf{W}$ having k columns, given $\mathbf{Y}$. We claimed that (i) 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 values; and (ii) the maximum value achieved is $\sum_{i=1}^{k} \sigma_i^2$.
Prove claims (i) and (ii).

3.1 Solution

We define the $m \times n$ matrix $\mathbf{Y}$ where $m \geq n$. Consider the singular value decomposition of the matrix $\mathbf{Y} = \mathbf{U\Sigma V^T}$. We now insert it into the expression that is to be maximized

$$tr(\mathbf{Y^T W W^T Y}) = tr(\mathbf{V\Sigma U^T W W^T U\Sigma V^T}) \tag{2}$$

Since for any matrix $\mathbf{A}$

$$tr(\mathbf{AA^T}) = tr(\mathbf{A^T A}) \tag{3}$$

we get that

$$tr(\mathbf{V\Sigma U^T W W^T U\Sigma V^T}) = tr(\mathbf{W^T U\Sigma V^T V\Sigma U^T W}) =$$

$$= tr(\mathbf{W^T U\Sigma I_{n\times n}\Sigma U^T W}) = tr(\mathbf{W^T U\Sigma\Sigma U^T W}) =$$

$$= tr(\mathbf{\Sigma U^T W W^T U\Sigma}) = \sum_{i=1}^{n} \sigma_i^2 \mathbf{u}_i^T \mathbf{W W}^T \mathbf{u}_i$$

$$= \sum_{i=1}^{n} \sigma_i^2 \left\| \mathbf{W}^T \mathbf{u}_i \right\|_2^2$$

where we have used the orthonormality of $\mathbf{V}$'s columns, equation (3) twice and the fact that the diagonal elements of $\mathbf{\Lambda}$ are σ_i. $\mathbf{u}_i$ are the columns of $\mathbf{U}$.

Since the columns of $\mathbf{W}$ are orthonormal and $\mathbf{U}$ is an orthonormal matrix by the definition of PCA and SVD respectively, we get that $\left\| \mathbf{W}^T \mathbf{u}_i \right\|_2^2 \leq 1 \ \forall i$ with an equality if and only if $\mathbf{u}_i$ is equal to one of the columns of $\mathbf{W}$. This is due to $\mathbf{a}^T\mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos(\alpha)$, which is maximized when $\alpha = 0$, where α is the angle between the two vectors. Since $\sigma_i^2 > 0 \ \forall i$ we want to maximize $\left\| \mathbf{W}^T \mathbf{u}_i \right\|_2^2 \ \forall i$ in order to maximize expression (2).

Due to $rank(\mathbf{W}) = k \leq n = rank(\mathbf{U})$ and their orthonormality, choosing one of $\mathbf{W}$'s columns to be equal to one of $\mathbf{U}$'s columns forces you to choose all of $\mathbf{W}$'s columns from $\mathbf{U}$'s columns in order for the assumptions of $\mathbf{W}$'s orthonormality to hold. The question is

now to decide which k columns to choose from $\mathbf{U}$.

σ_i^2 is decreasing in i so we should choose the k first columns of $\mathbf{U}$ corresponding to the k largest singular values, which can be denoted $\mathbf{U}_k$. Thus, $\mathbf{W} = \mathbf{U}_k$. It is now easy to find that the maximum value is

$$\sum_{i=1}^{n} \sigma_i^2 \left\| \mathbf{W}^T \mathbf{u}_i \right\|_2^2 = \sum_{i=1}^{n} \sigma_i^2 \left\| \mathbf{U}_k^T \mathbf{u}_i \right\|_2^2 = \sum_{i=1}^{k} \sigma_i^2$$

where the last equality is due to the orthonormality of $\mathbf{U}$'s columns.

■

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 $s_{ij} = -\frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$.

We claim that s_{ij} can be computed by $s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$, where d_{1i} and d_{1j} are the distance 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 $\mathbf{S}$.

4.1 Solution

Note that $d_{ij}^2 = \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 = (\mathbf{y}_i - \mathbf{y}_j)^T(\mathbf{y}_i - \mathbf{y}_j) = \mathbf{y}_i^T\mathbf{y}_i - 2\mathbf{y}_i^T\mathbf{y}_j + \mathbf{y}_j^T\mathbf{y}_j$.

Extending the estimation of s_{ij} we get

$$-\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2) = -\frac{1}{2}((\mathbf{y}_i^T\mathbf{y}_i - 2\mathbf{y}_i^T\mathbf{y}_j + \mathbf{y}_j^T\mathbf{y}_j) - (\mathbf{y}_1^T\mathbf{y}_1 - 2\mathbf{y}_1^T\mathbf{y}_i + \mathbf{y}_i^T\mathbf{y}_i) - (\mathbf{y}_1^T\mathbf{y}_1 - 2\mathbf{y}_1^T\mathbf{y}_j + \mathbf{y}_j^T\mathbf{y}_j)) =$$

$$= -\frac{1}{2}(-2\mathbf{y}_i^T\mathbf{y}_j - 2\mathbf{y}_1^T\mathbf{y}_1 + 2\mathbf{y}_1^T\mathbf{y}_i + 2\mathbf{y}_1^T\mathbf{y}_j) = \mathbf{y}_i^T\mathbf{y}_j + \mathbf{y}_1^T\mathbf{y}_1 - \mathbf{y}_1^T\mathbf{y}_i - \mathbf{y}_1^T\mathbf{y}_j$$

$$= (\mathbf{y}_i - \mathbf{y}_1)^T(\mathbf{y}_j - \mathbf{y}_1)$$

which in matrix form can be expressed as

$$\mathbf{Y}^T\mathbf{Y} + \mathbf{1}_n\mathbf{1}_n^T\mathbf{y}_1^T\mathbf{y}_1 - \mathbf{1}_n\mathbf{y}_1^T\mathbf{Y} - \mathbf{Y}^T\mathbf{y}_1\mathbf{1}_n^T = (\mathbf{Y} - \mathbf{y}_1\mathbf{1}_n^T)^T(\mathbf{Y} - \mathbf{y}_1\mathbf{1}_n^T)$$

This is clearly a version of the similarity matrix where $\mathbf{Y}$ has been transformed by subtracting y_1 from every datapoint, that is centered to y_1. The centering won't affect the distances between the individual points since

$$\|(\mathbf{y}_i - \mathbf{y}_1) - (\mathbf{y}_j - \mathbf{y}_1)\|_2 = \|\mathbf{y}_i - \mathbf{y}_j\|_2 = d_{ij}$$

We thus see that this is one of the possible estimations of the Gram matrix $\mathbf{S}$.

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 terms of computation, which is the best way to perform the embedding?

5.1 Solution

Like PCA, metric MDS relies on a simple generative model. More precisely, only an orthogonal axis change separates the observed variables in y and the latent ones, stored in $\mathbf{x}$:

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

where $\mathbf{W}$ is a $D \times P$ matrix such that $\mathbf{W}^T\mathbf{W} = \mathbf{I}_P$. This means that the similarity matrix can be expressed as follows:

$$\mathbf{S} = \mathbf{Y}^T\mathbf{Y} = (\mathbf{W}\mathbf{X})^T(\mathbf{W}\mathbf{X}) = \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X} = \mathbf{X}^T\mathbf{X}$$

Using the singular value decomposition of $\mathbf{Y}$, $\mathbf{Y} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T$ (note that $\mathbf{U}$ and $\mathbf{V}$ are switched, it will make the proof easier), one can express the similarity matrix as follows:

$$\mathbf{S} = \mathbf{Y}^T\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{U}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{U}^T = \mathbf{U}\mathbf{\Lambda}_{MDS}\mathbf{U}^T$$

where $\mathbf{\Lambda}_{MDS} = \mathbf{\Sigma}\mathbf{\Sigma}^T$. The solution is $\hat{\mathbf{X}}_{MDS} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T$ since

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}_{MDS}\mathbf{U}^T = (\mathbf{\Lambda}^{1/2}\mathbf{U})^T\mathbf{U}^T\mathbf{\Lambda}^{1/2} = \mathbf{X}^T\mathbf{X}$$

Using the results from Problem 3, that $\hat{\mathbf{X}}_{PCA} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{Y}$ (here $\mathbf{I}_{P\times D}\mathbf{V}^T = \mathbf{V}_D^T$, where $\mathbf{V}$ and D are equivalent to $\mathbf{U}$ and k respectively), we get that

$$\hat{\mathbf{X}}_{MDS} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T = \mathbf{I}_{P\times N}(\mathbf{\Sigma}^T\mathbf{\Sigma})^{1/2}\mathbf{U}^T = \mathbf{I}_{P\times D}\mathbf{\Sigma}\mathbf{U}^T$$

and

$$\hat{\mathbf{X}}_{PCA} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{Y} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}^T = \mathbf{I}_{P\times D}\mathbf{\Sigma}\mathbf{U}^T$$

where we have used the orthonormality of $\mathbf{V}$ and the SVD of $\mathbf{Y}$.

Computation wise, the two methods are better in different situations. When the data are not too high-dimensional $\mathbf{Y}\mathbf{Y}^T$ is smaller in size than $\mathbf{Y}^T\mathbf{Y}$, which means that PCA would spend fewer memory resources than MDS. In the opposite case, when the dimensionality

is very high but the number of points rather low, MDS would to better due to the same reason.

■

6 Problem 6

Argue that the process to obtain the neiborhood graph G in the Isomap method may yield a disconnected graph. Propose a heuristic to patch this problem. Justify your heuristic.

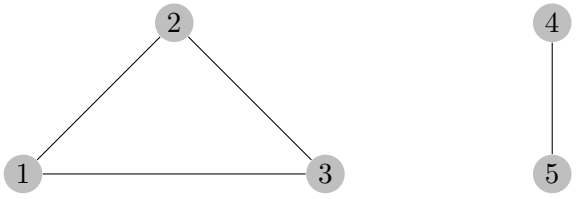
6.1 Solution

A heuristic to this problem is:

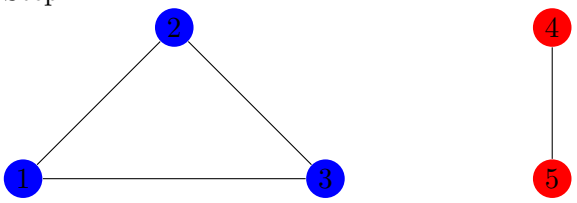
1. Identify all disconnected graphs
2. Calculate the centers of all disconnected graphs
3. Connect the calculated centers with each other and to all vertices in their respective subgraphs
4. Scale all new edges between the disconnected graphs with a number $c \geq 1$
5. Tune c to get the best solution

This approach will lead to a connected graph. It makes sense since only creates a single edge per pair of disconnected graphs, thus not increasing the complexity of the graph significantly. The purpose of step 4 and 5 may not seem obvious. The edges between the new edges between the centers should be scaled with a number $c \geq 1$ since distance between the disconnected graphs had to be large for them to be disconnected from the beginning (if k isn't very small). These long euclidean distances we now calculate would therefore not be good approximations of the geodesic distances since the distance on the manifold would be much larger than the "straight" euclidean distance. Finally, c has to be tuned since the high-dimensional manifold will vary in form from graph to graph.

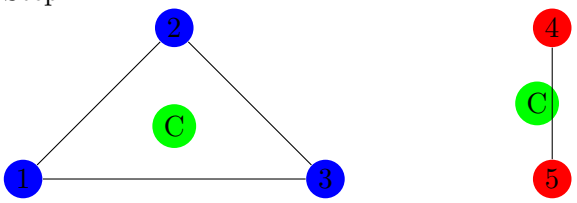
Following is an example of the algorithm being run on a small graph with five vertices. In the final two step the edge between the two centers were scaled with c and c^* respectively. The center of the second graph is not perfectly centered for illustratory reasons.



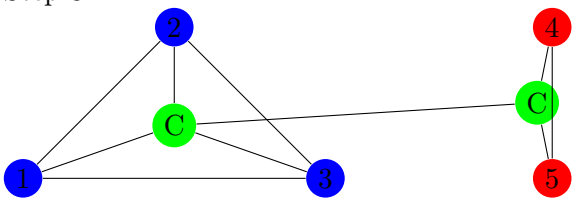
Step 1:



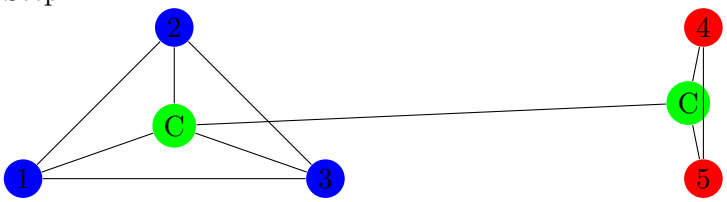
Step 2:



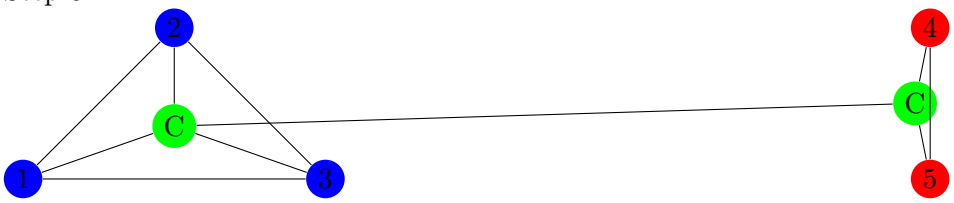
Step 3:



Step 4:



Step 5:



7 Problem 7

We want to visualize a small set of animal species. We will use the "zoo" dataset in the UCI ML repository: https://archive.ics.uci.edu/ml/datasets/zoo, which contains information about a few different attributes of each species. We want to project the data in 2D so that "similar" animals are projected near to each other. The last attribute "type" can be removed from the dataset, and used as color for each point in the visualization. Since the dataset is fairly small, you may also want to annotate each point in the projection.

Note that all attributes are Boolean, except one, so first you will need to decide how to handle it.

You are asked to experiment with the following methods:

1. PCA.
2. MDS, where you can try to infer the importance of different attributes, compute pairwise distances between the species taking into account the attribute importance, and apply MDS on the resulting distance matrix.
3. Isomap, where you should experiment with the number of nearest-neighbor parameter used to form the neighborhood graph.

Describe what you have implemented and justify the choices you have made. Plot your results. Write down your observations and conclusions. Which method is preferable?

7.1 Solution

The dataset used for this problem contains information about animal species. It consists of 101 datapoints (animal species) and 17 attributes. The last attribute "type" ($\in \{1, \ldots, 7\}$) will only be used for coloring each point in the visualization. All other attributes are Boolean (they will be handled as $\{0, 1\}$), except "legs", which takes the values in $\{0, 2, 4, 5, 6, 8\}$. All data points will be annotated with their respective animal species followed by their animal type.

The goal is to use three different techniques to project the 16D data in 2D such that animals of different types are projected close to each other. The programming language used was Python.

Normally, it would be wise to scale the variables when they have different units (in this case legs not being Boolean), especially for PCA. However, in this case, the attribute "legs" showed to be a great predictor, so it was kept on its ordinary scale to not inhibit its influence on the results.

The code can be found in the following repository:
https://gits-15.sys.kth.se/aregnell/DD2434_Assigment_1

7.1.1 PCA

We want to perform PCA on the data $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_{100}) \in \mathbb{R}^{16\times 100}$ in order to project it in 2D. Thus, we start of by calculating the singular value decomposition of $\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and then choose the k columns of $\mathbf{U}$ corresponding to the largest singular values of $\mathbf{\Sigma}$ and put them in the matrix $\mathbf{U}_k$. The projection is then done as follows:

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

where $\mathbf{X} \in \mathbb{R}^{2\times 100}$, since we chose $k = 2$. This was done by using the **sklearn.decomposition** library. The result of the dimensionality reduction achieved by PCA can be seen in figure 1.

Looking at figure 1 we see that PCA does well on the blue and purple points. However, it does have a hard time differentiating the lime green, the light green and the turquoise points, putting them in two clusters with the turquoise points split in between them. The same is true with the orange and red points. To summarize, PCA performed adequately.

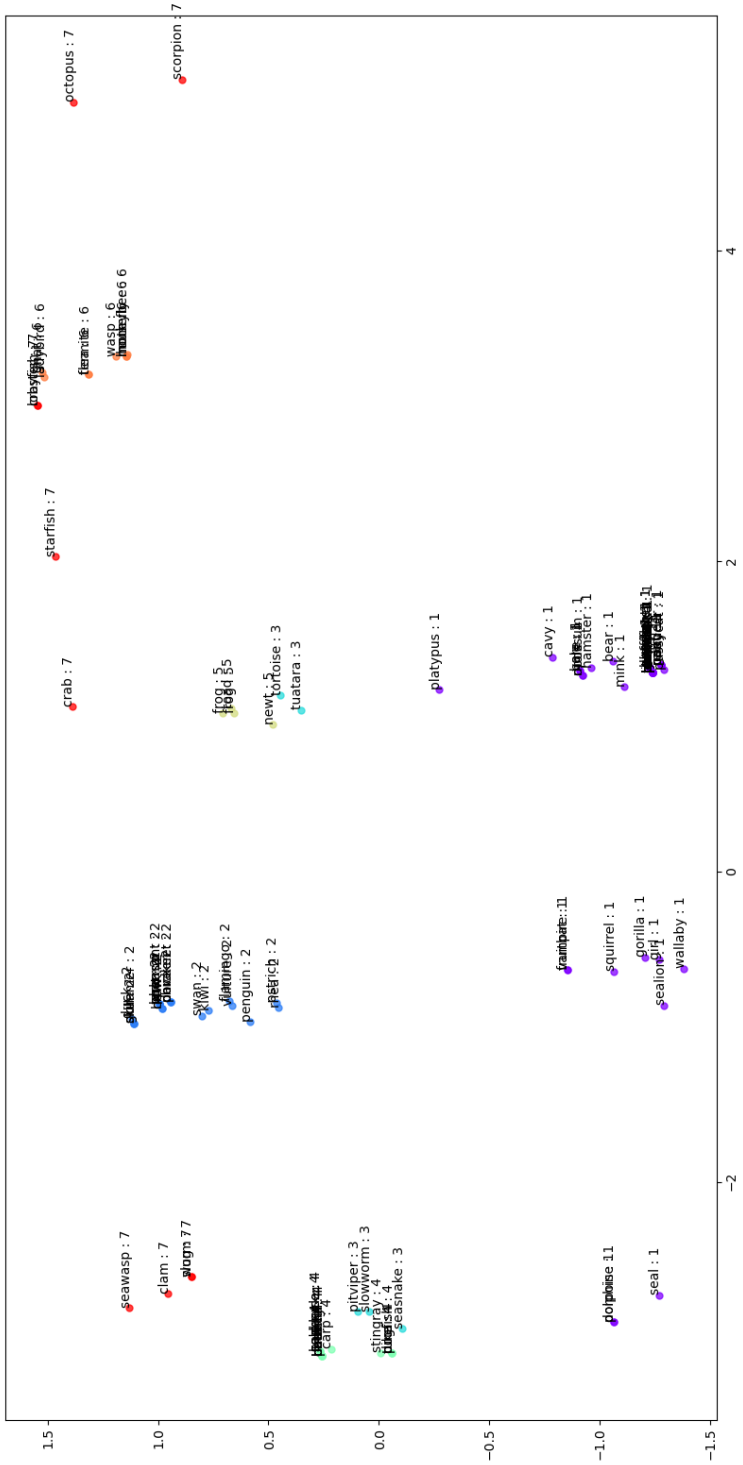


Figure 1: The data projected in 2D with PCA.

7.1.2 MDS

Multidimensional scaling, MDS, is very closely related to PCA. In fact, as shown in Problem 5, the result PCA and MDS procedures are equivalent. To perform MDS one first calculates the simlarity matrix $\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$. It is followed by an eigen-decomposition of $\mathbf{S}$, $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. The projection is then done as follows:

$$\mathbf{X} = \mathbf{I}_k\mathbf{\Lambda}^{1/2}\mathbf{U}^T$$

where $\mathbf{X} \in \mathbb{R}^{2\times 100}$, since we once again chose $k = 2$.

In many cases one is given the distance matrix $\mathbf{D}$, where $[\mathbf{D}]_{ij} = d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$, and not the similarity matrix. However, the similarity matrix can be estimated by a "double centering" trick (and in other ways, see Problem 4), which in matrix form looks as follows:

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

In this problem we were actually given $\mathbf{Y}$ and could thus calculate the similarity matrix directly. However, we were asked to first calculate distance matrix and then perform MDS on it, which is what have been done. First, the distance matrix was calculated by $d_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$ and second, then the similarity matrix was calculated with the double centering trick. The eigen-decomposition was performed with the **numpy.linalg** library. The results can be seen in figure 2.

One clearly sees that this is the same projection PCA achieved, which was expected. However, both axes have been flipped, probably explained by the two different libraries being used. One should also note that this flip is not "wrong", since if $\mathbf{u}$ is an eigen-vector with eigen-value λ, so is $-\mathbf{u}$. This means that MDS, as PCA, also performed adequately.

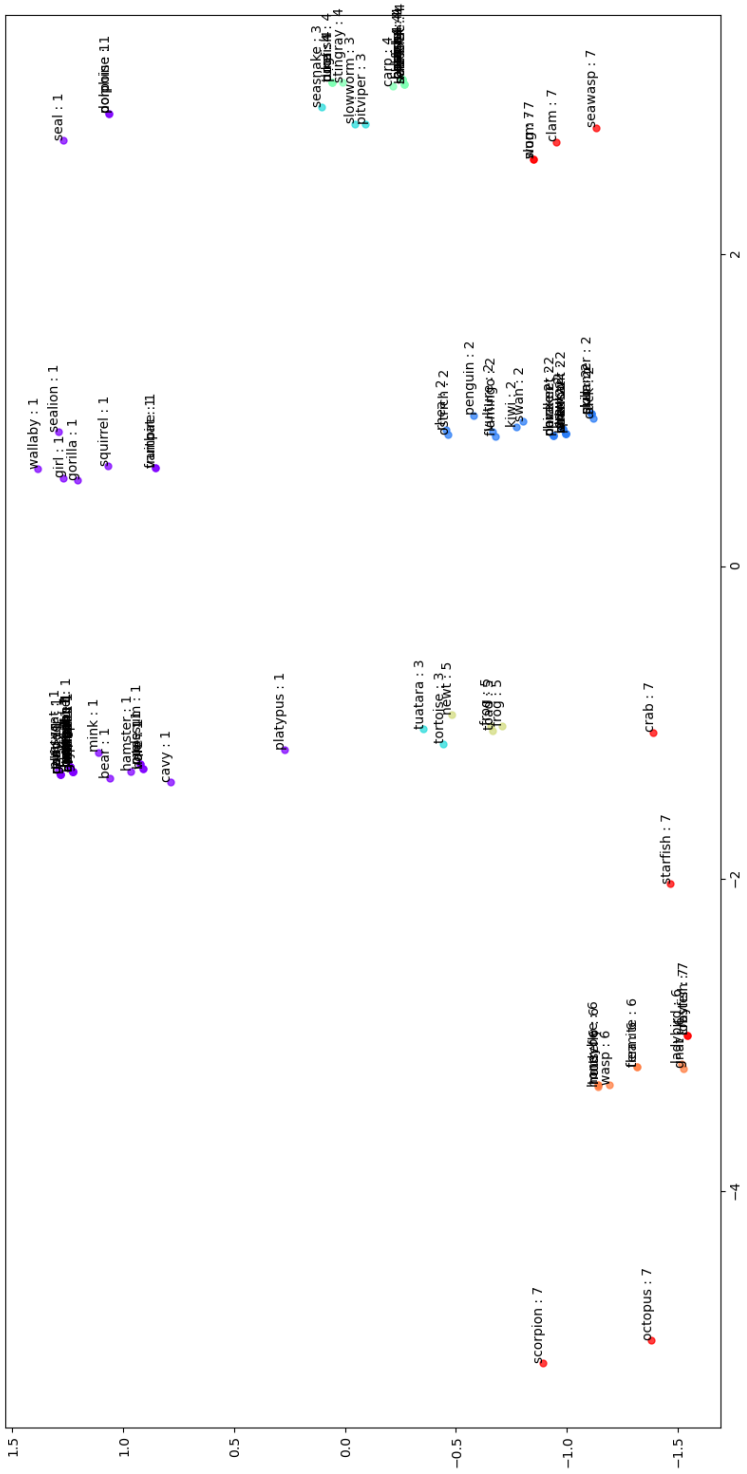


Figure 2: The data projected in 2D with MDS.

Entropy gain	13	4	8	3	1	2	9	10	14	5	12	6	16	11	7	15
feature importance	4	3	2	13	8	1	9	10	12	14	5	6	7	16	11	15
Permutation	3	9	6	4	10	16	15	7	2	13	12	14	5	8	1	11

Table 1: Importance of the attributes 1-16, from most to least important.

Attribute	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Entropy gain	0.29	0.27	0.31	0.36	0.18	0.14	0.04	0.32	0.25	0.23	0.05	0.18	0.51	0.19	0.02	0.11
feature_importance_	0.26	0.38	0.41	0.43	0.12	0.14	0.04	0.29	0.25	0.29	0.06	0.23	0.29	0.18	0.0	0.06
Permutation	0.23	0.25	0.27	0.26	0.23	0.27	0.27	0.22	0.28	0.25	0.2	0.23	0.24	0.24	0.27	0.26

Table 2: Weights given to the attributes 1-16 based on their importance.

The instructions also asked for us to infer the importance of the different attributes and use it when calculating the pairwise distances between the species. This was done using three different methods.

The first method was is based on calculating the entropy gain of the different attributes. This was done by splitting the dataset with respect to the values a particular attribute takes. This was done for every attribute using the following formula:

$$\text{Gain}_i = \text{Ent}(\mathbf{Y}) - \sum_{v \in \text{Values}(A)} \frac{\mid \mathbf{Y}_v \mid}{\mid \mathbf{Y} \mid} \text{Ent}(\mathbf{Y}_v)$$

where A is the attribute, $\mid \mathbf{Y} \mid$ the number of points in dataset $\mathbf{Y}$ and $\mathbf{Y}_v$ the subset of $\mathbf{Y}$ where attribute A take the value v. $\text{Ent}(\mathbf{Y})$ is the Shannon entropy of $\mathbf{Y}$.

The entropy gains were saved in a weight vector $\mathbf{w} = (\text{Gain}_1, \ldots, \text{Gain}_{16})$.

The second method fitted a regression tree to the data using the **sklearn.ensemble** library and used the attributes **.feature_importances_** to get the importance of the 16 attributes. These were also saved in a weight vector.

The third method also fitted a regression tree to the data in the same way as the second method. However, to infer the attribute importance it used a permutation feature importance technique. It works by making one of the attributes unusable (by shuffling it) and then measuring how the model performs. The attributes for which the model performs worst when it is shuffled should be the most important one.

The decrease in accuracy of the regression tree on the test data for when the respective attributes were shuffled was also saved into a weight vector.

All three weight vectors were normalized to make the comparison easier. Now, every dimension was scaled with their respective weight in the weight vector (one method at a time). This should have the effect that the more "important" dimensions will have a greater impact on the pairwise distances in the distance matrix.

Looking at table 1 and 2 we see that the entropy gain and feature importance method

give quite similar results, both in order (e.g. attribute 4 was rated second and most important respectively) and magnitude of weights (ranging from 0 to 0.5). These methods are thus expected to make a noticeable difference. However, the permutation method had almost no similarities in the order and the weights were almost invariant, ranging from 0.2 to 0.28. This method is thus expected to give similar results to the ordinary MDS.

The result of the three different scaling methods can be seen in figure 3, 4 and 5 respectively.

Comparing figure 3 to 4 we see that they are almost identical, with the exception the feature importance method gave slightly tighter clusters for the purple and blue data points. Compared to the ordinary MDS the clusters are much tighter, but it still has some trouble with the red, orange, light green, lime green and turquoise points. One also notices that the axes have been flipped again, as they were for PCA. The permutation method gave almost the exact same results as the ordinary MDS, as was expected, however, now only the y-axis was flipped.

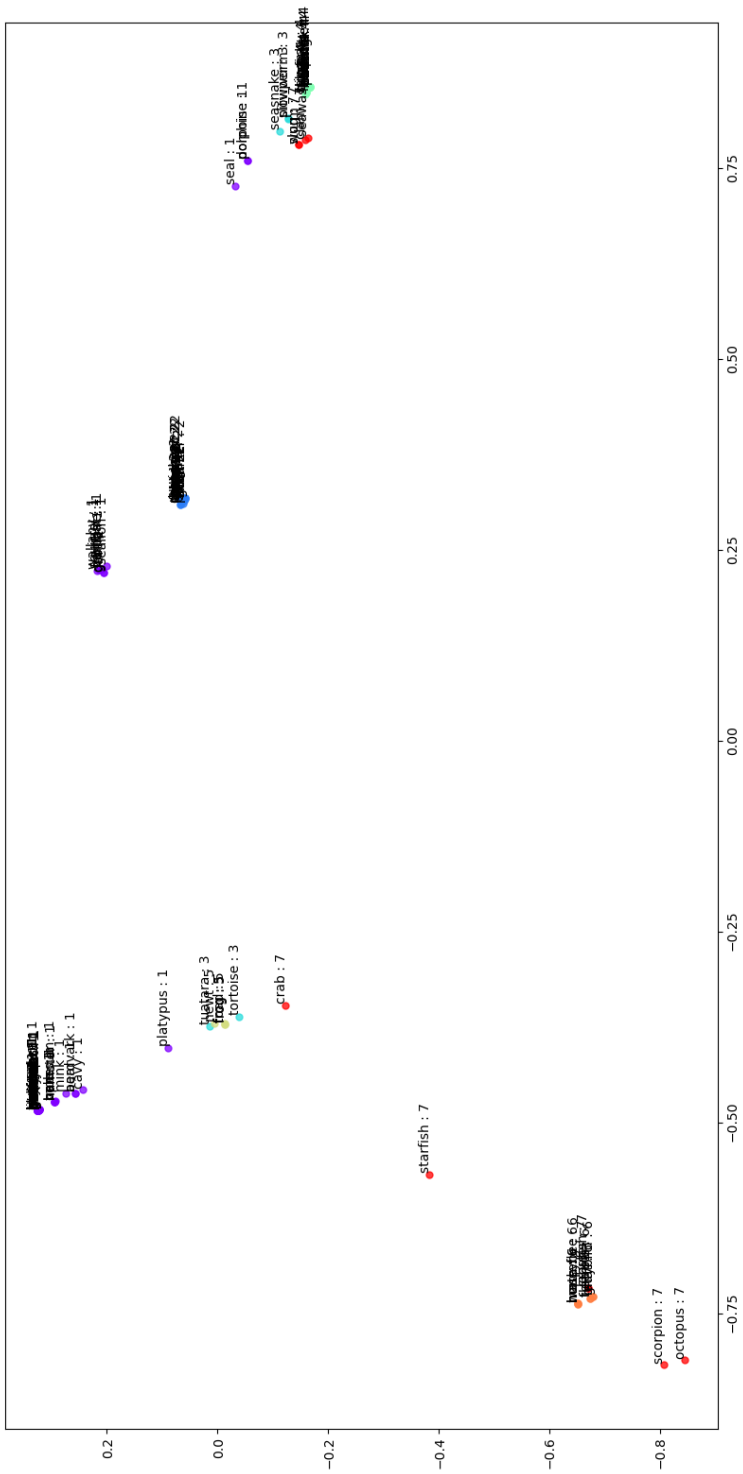


Figure 3: The data projected in 2D with MDS and entropy gain scaling.

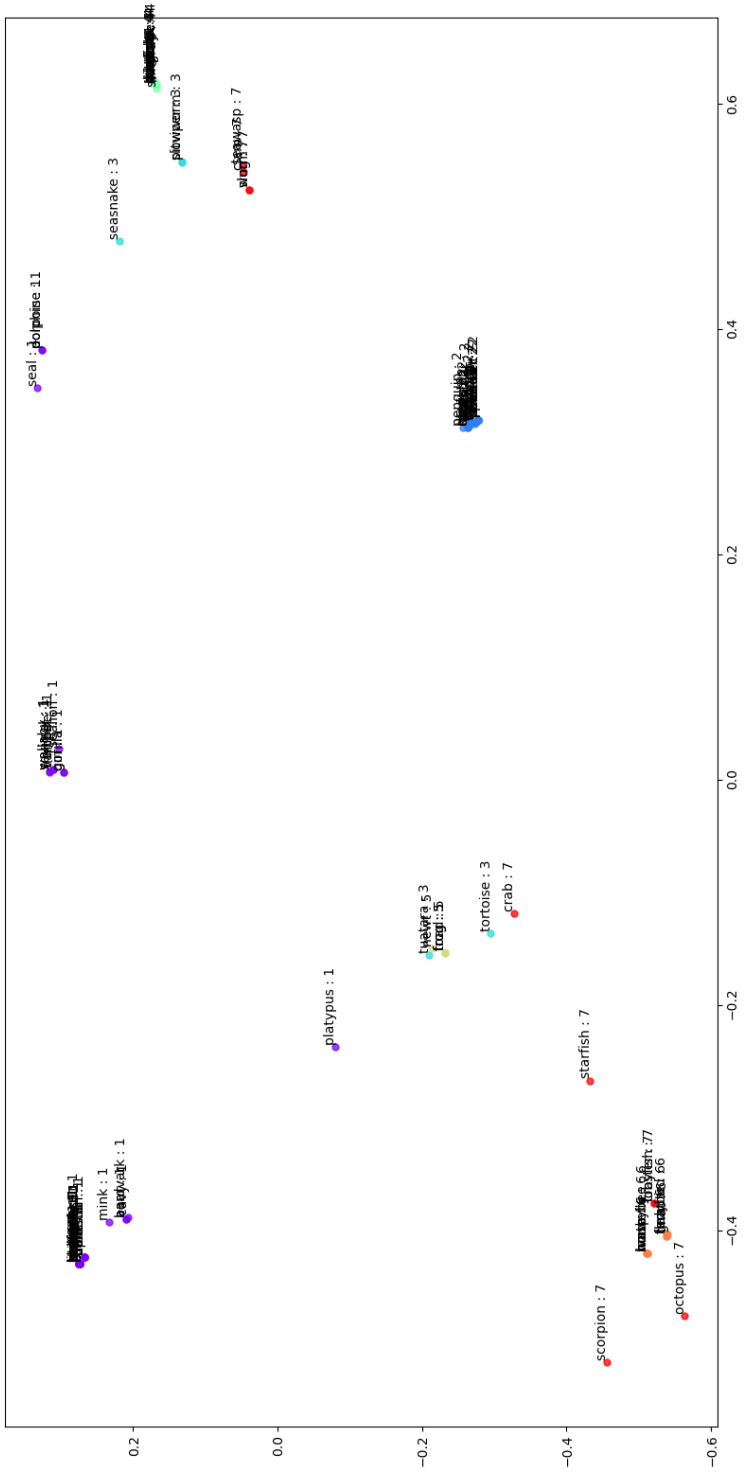


Figure 4: The data projected in 2D with MDS and regression tree **.feature importances**.

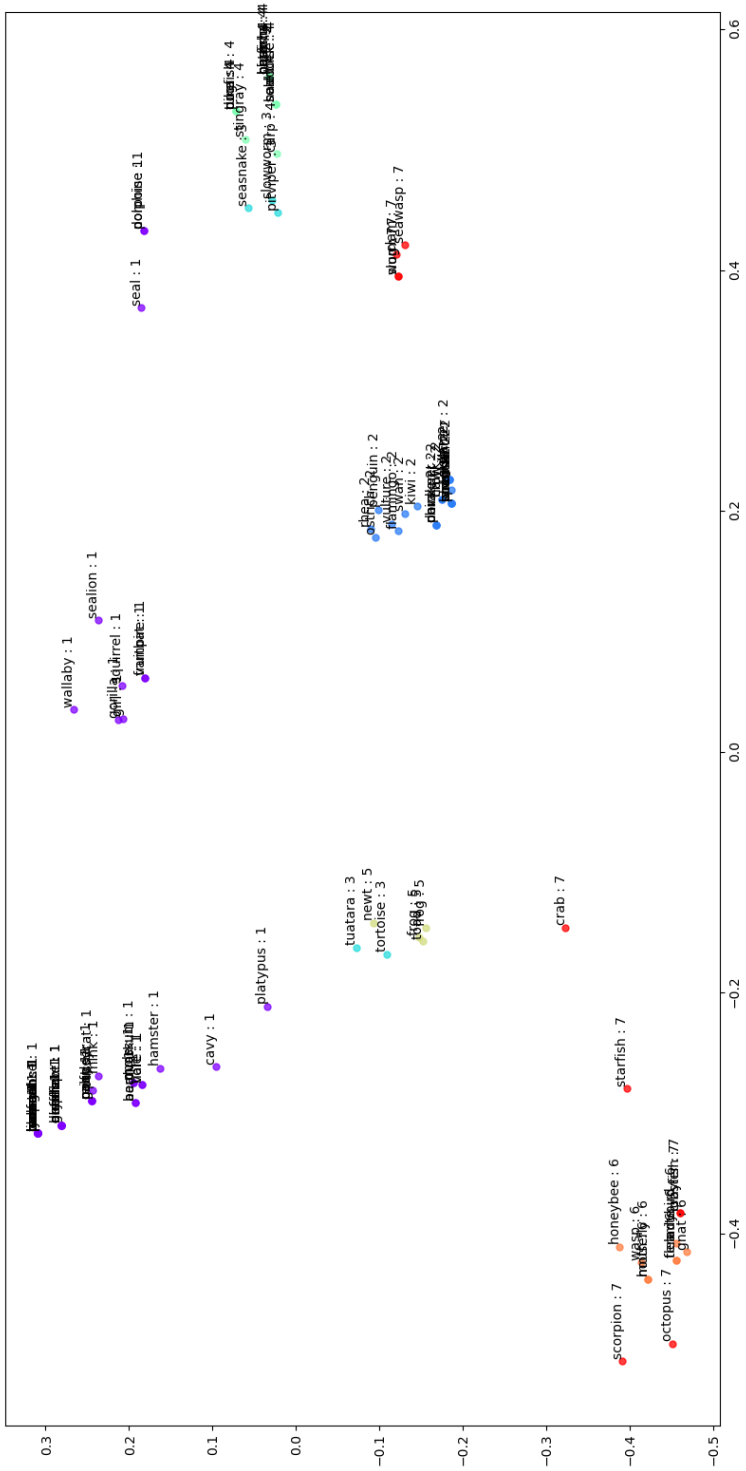


Figure 5: The data projected in 2D with MDS and regression tree permutation importance.

7.1.3 Isomap

The isomap method tries to combat the problems with MDS and PCA. Datasets usually lie on high-dimensional manifolds and thus euclidean distance can be a poor mesure of dissimilarity between points. Instead one should use the geodesic distance, that is the distance between points on the manifold, since it better captures the neighborhood relationship between the points.

Calculating the geodesic distances is impossible when the manifold is unknown, but it can be approximated by assuming that the geodesic distance is close to the Euclidean distance for nearby points. The geodesic distance between faraway points can then be approximated by sequence of "short hops" between neighboring points.

This can be achieved by the following algorithm:

1. Construct a graph G, where vertices represent points, and each point is connected to its k nearest point, according to their pairwise squared Euclidean distances in the original space. The edges are scaled with the squared Euclidean distance between the vertices it connects to.

2. For each pair of points i and j compute the shortest path distance d_{ij} from i to j on the graph G.

3. Use MDS on the shortest-path distance matrix $\mathbf{D}$.

This was achieved by using the **kneighbors_graph** function from the library **sklearn.neighbors** to calculate G and then using Floyd-Warshall algorithm to compute the shortest path distance matrix $\mathbf{D}$ of G. Finally, the same MDS implementation as above was used on the shortest path distance matrix $\mathbf{D}$.

The smallest k for which the neighborhood graph G was not disconnect was $k = 28$. The projection it achieved can be seen in figure 6.

By increasing k the "short hops" get longer since more points are considered as neighbors. When increasing k, estimations of the geodesic distances should become more accurate in the beginning since there will be less "zig zag hops" within the manifold. However, after a while the increased distances of the hops will lead to more "shortcuts" outside the manifold, decreasing the accuracy of the geoedesic distances. Finally, when $k = 100$, all points are neighbors to each other and thus the neighborhood graph converges with the distance matrix (as in MDS) except that every element is squared since the shortest path between two points will be their pairwise squared Euclidean distance.[1] Thus there should exist an optimal $\hat{k}$ such that $28 \leq \hat{k} \leq 100$. The Isomap projection for other values of k can be seen in figure 7 to 11.

Like MDS and PCA, Isomap is also good at keeping the purple and blue points close together, but not as tight as the MDS with the first to weighting methods. It also had

[1] This shows that Isomap is related to both PCA and MDS.

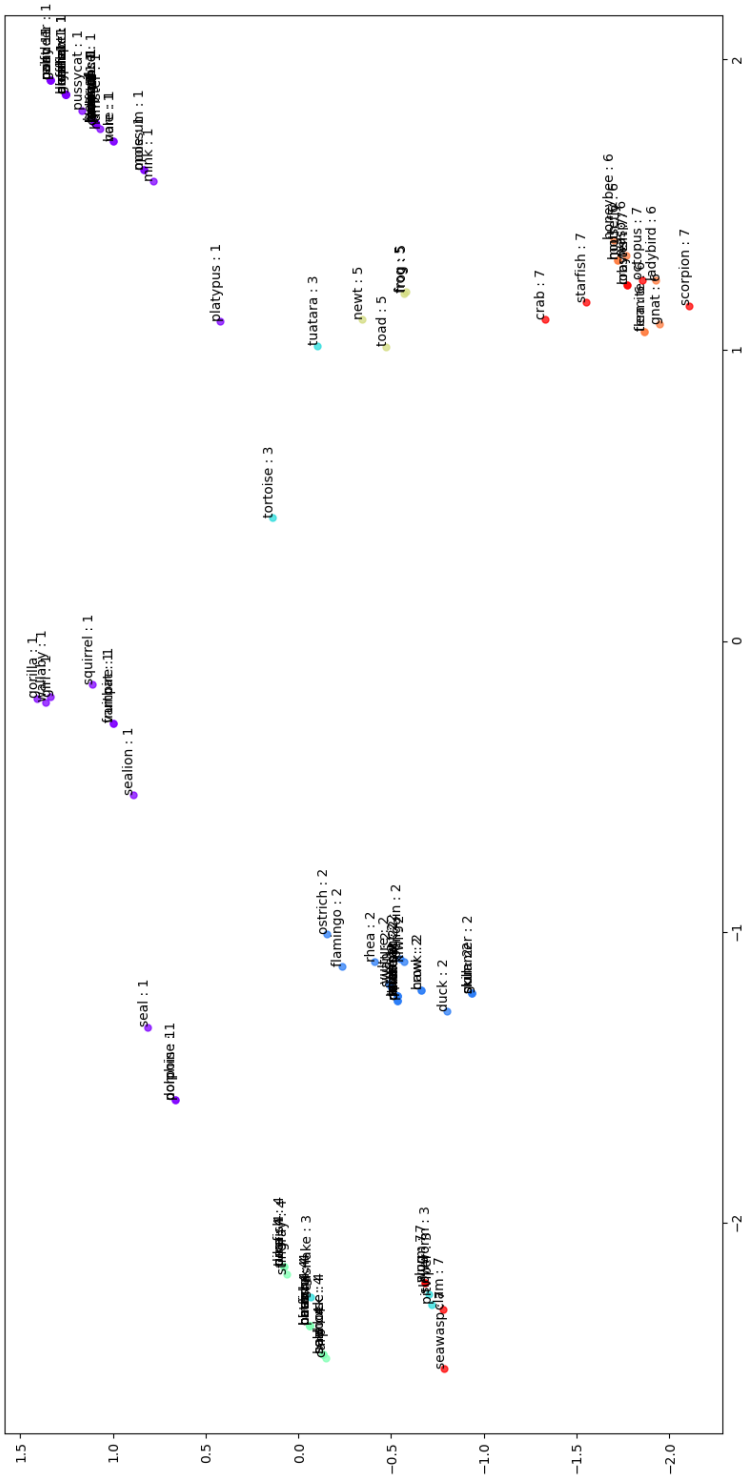


Figure 6: The data projected in 2D with Isomap with 28 nearest neighbors.

the same problems of separating the lime green, the light green and the turquoise points as well as separating the red from the orange points. However, for $k \in \{40, 60, 80\}$ it kept the orange points in a relatively tight cluster. It did the same for the red points, albeit in two separate clusters.

As a final test, I also tried how the methods used in the MDS part would affect the projection when they were applied on the geodesic distance matrix. The results of the case when $k = 38$ and the method used was the feature importance can be seen in figure 12. One sees that it performs well for the blue points but compared to the other methods it does better on the lime green points and worse on the purple points.

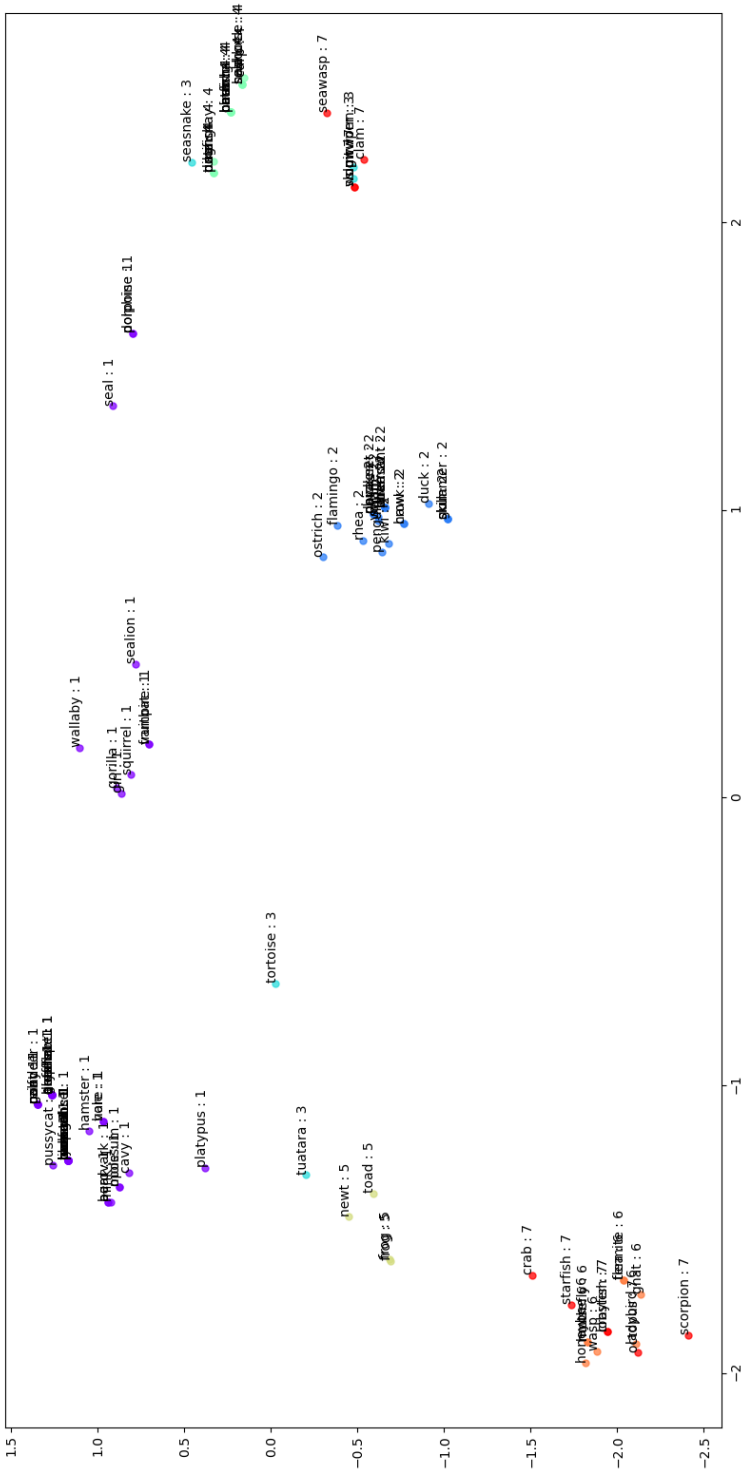


Figure 7: The data projected in 2D with Isomap with 32 nearest neighbors.

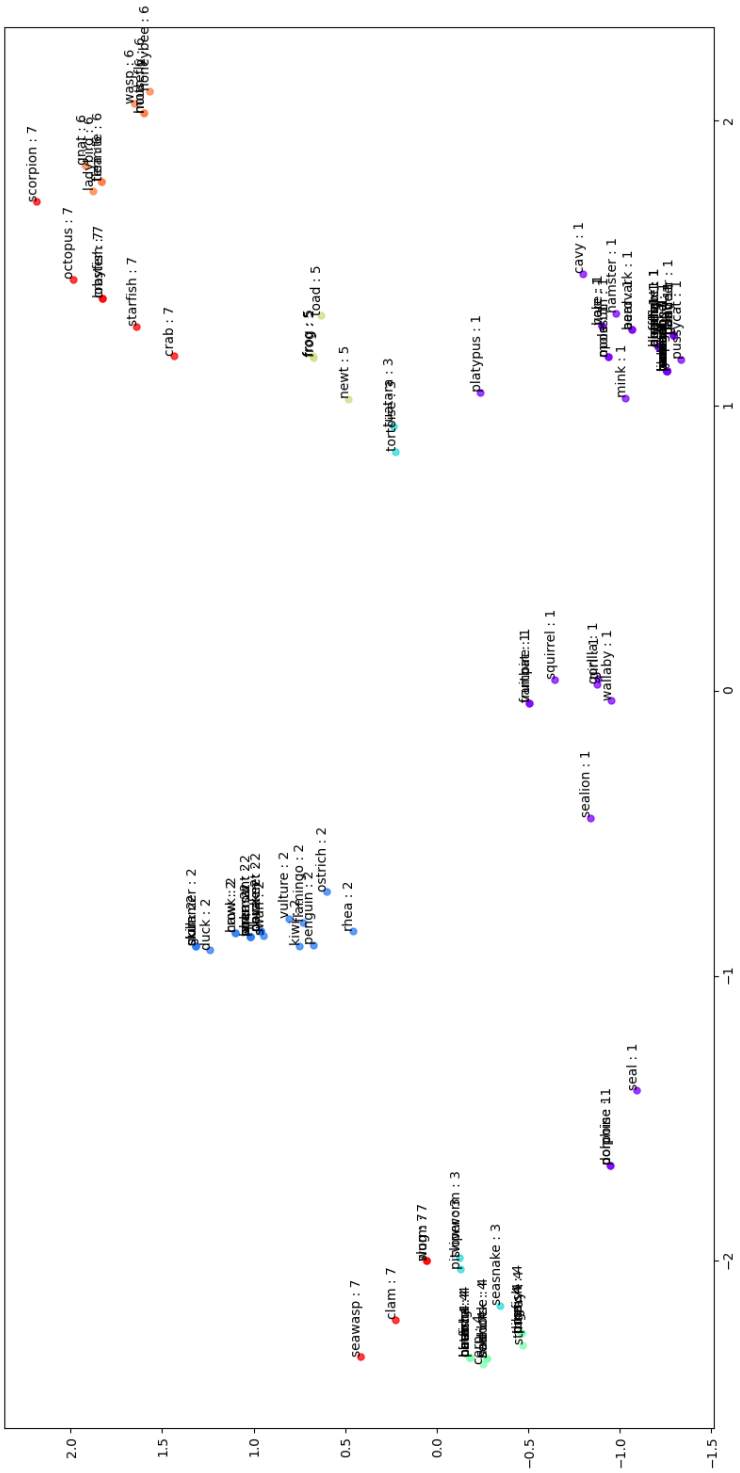


Figure 8: The data projected in 2D with Isomap with 40 nearest neighbors.

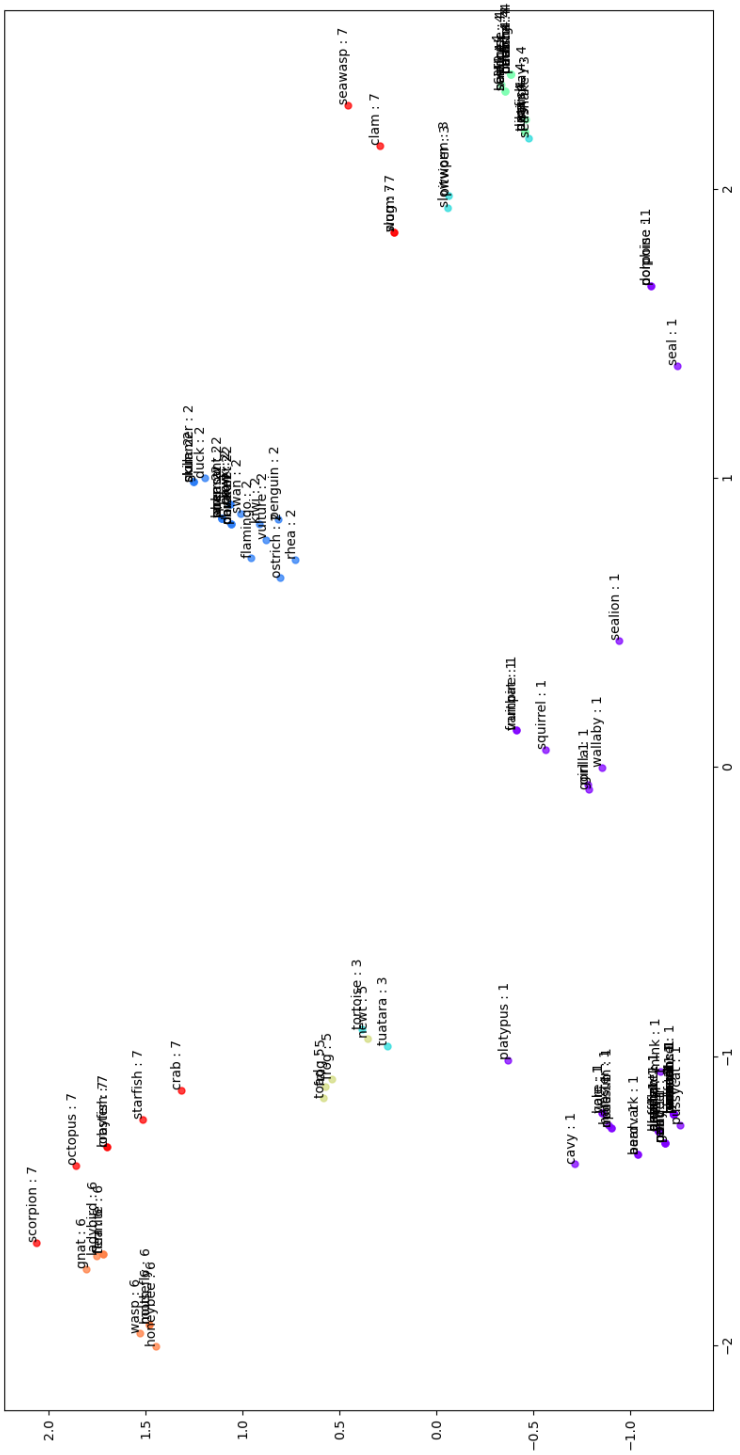


Figure 9: The data projected in 2D with Isomap with 60 nearest neighbors.

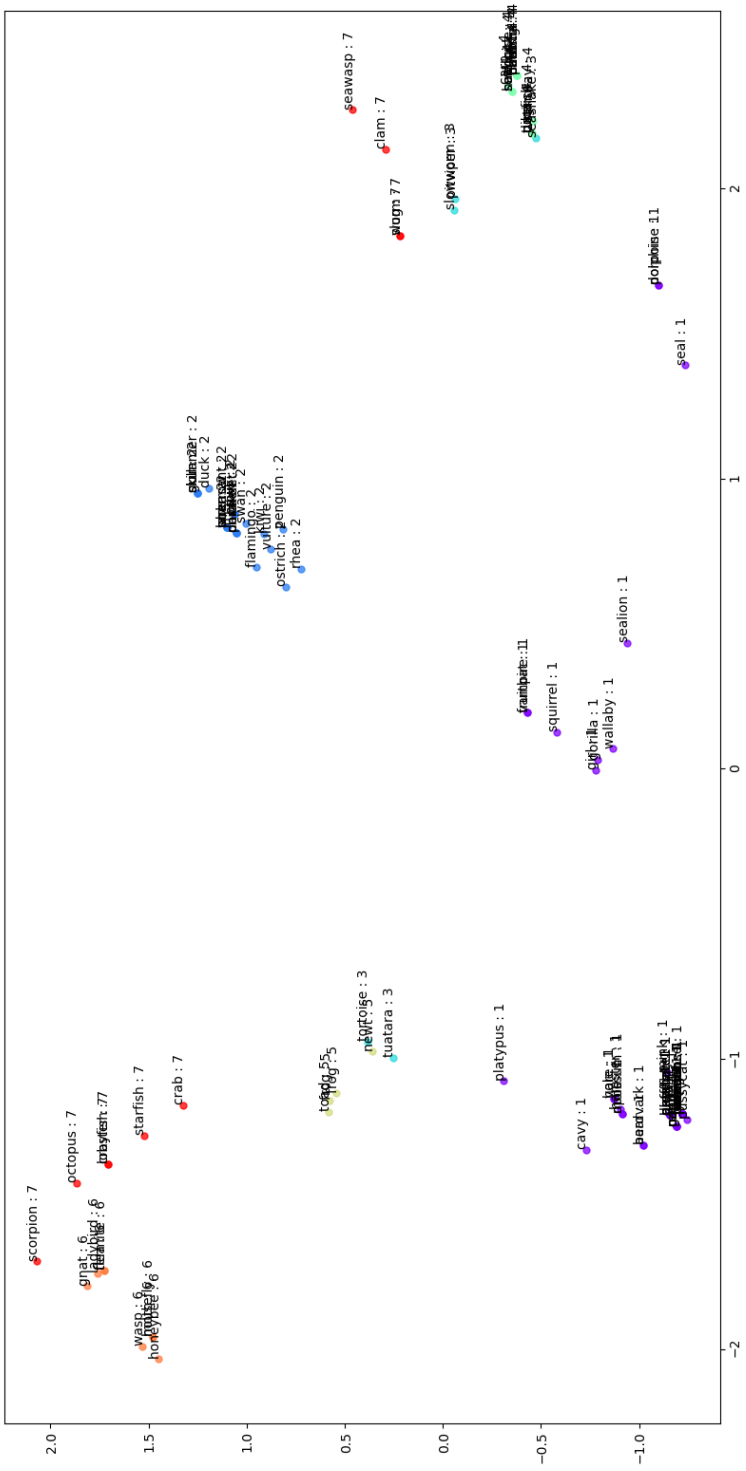


Figure 10: The data projected in 2D with Isomap with 80 nearest neighbors.

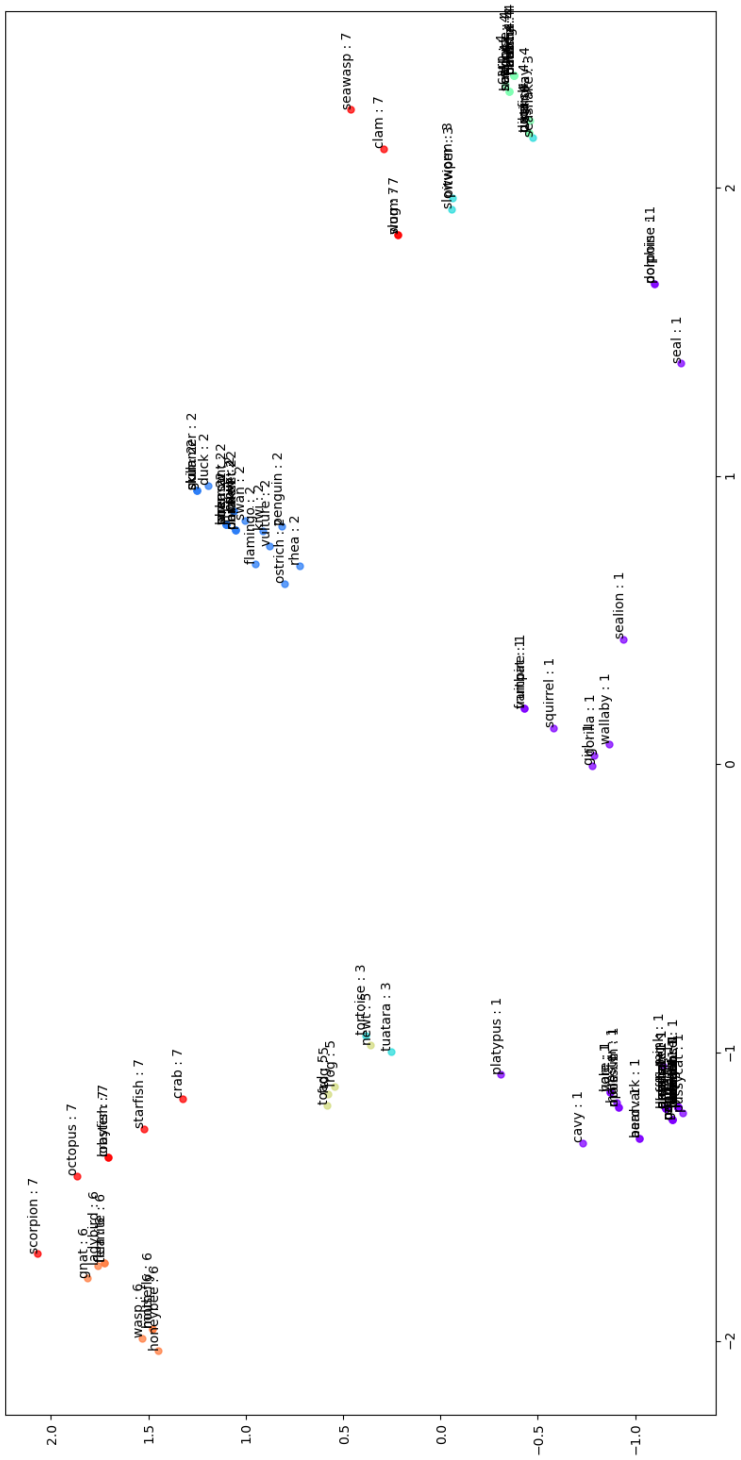


Figure 11: The data projected in 2D with Isomap with 100 nearest neighbors.

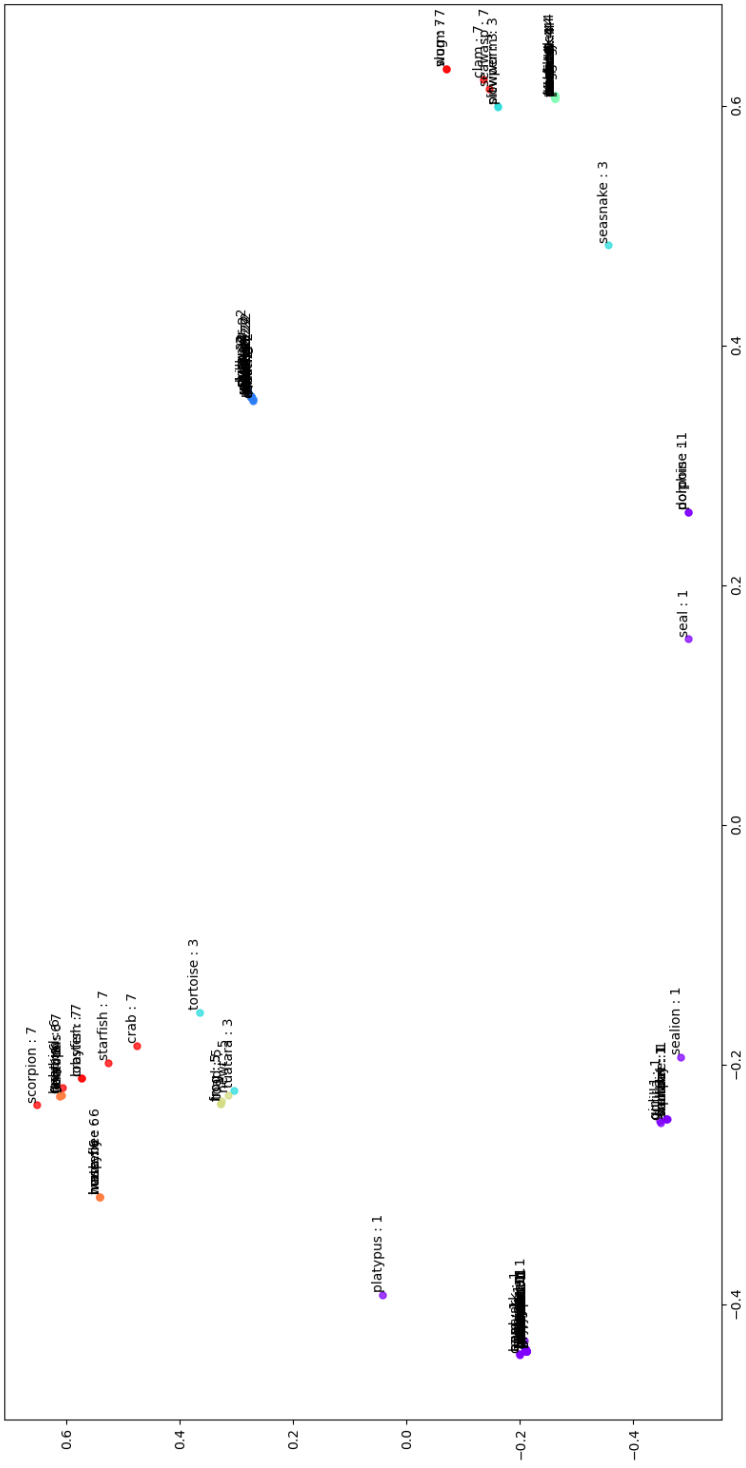


Figure 12: The data projected in 2D with Isomap with 38 nearest neighbors. Scaling of the distance matrix was done with the feature importance method.

7.1.4 Summary and Conclusions

Comparing all models one finds two clear winners, MDS and Isomap ($k = 38$), both with the feature importance. Both had really tight clusters for the lime green, the blue and the purple data points, however, one could give a slight edge to the latter since it gave a tighter cluster for the (some of) red and orange points. The worst performers were the PCA and MDS (ordinary and permutation weights), they managed to cluster the similar species but they were sparser than the other methods.

For all PCA and MDS methods one sees that one of their singular vectors were heavily influenced by the "legs" variable since all point lie on six different lines. This phenomena is not as obvious for the Isomaps, indicating that it takes more variables into account.

These results are not unexpected, PCA and MDS are designed to handle only linear submanifolds, while Isomap can handle a much wider class of manifolds. Isomap can reduce dimensionality of linear and developable nonlinear manifolds without any loss. However, Isomap cannot handle all types of manifolds, and when it is met with a nondevelopable manifold, it suffers from the same limitations as PCA or metric MDS applied to a nonlinear manifold. It also sensitive to the choice of k, which was discussed earlier.[2]

[2] John A. Lee & Michel Verleysen, *Nonlinear Dimensionality Reduction*, page 110