DATA 604: Data Representation and modelling

Final Project

Sweta Dantu
UID: 120429101

Abstract

This project aims to see how different methods of representing data affect classification accuracy, using the dataset Animal-10. It compares raw data, Principal Component Analysis (PCA), kernel PCA (kPCA), and Locally Linear Embedding (LLE) for data representation, alongside classifiers like k-Nearest Neighbors (kNN). The project explores data preprocessing methods, comparison techniques, and classifier performance. Its aim is to offer insights into the most effective methods for data representation and classification.

1 Introduction

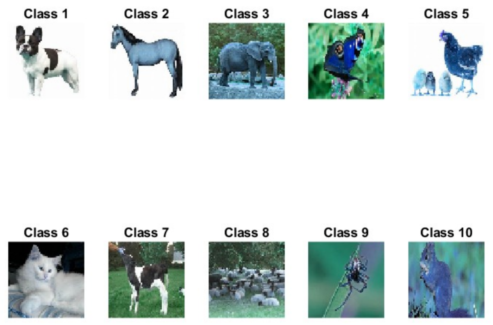
The data set used in this project is Animal-10 found on Kaggle. This dataset contains about 28K medium quality animal images belonging to 10 categories: dog, cat, horse, spyder, butterfly, chicken, sheep, cow, squirrel, elephant. Image count for each animal(category) varies from 2K to 5K images.

2 Data pre-processing

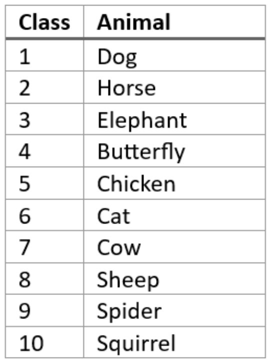
The images in the dataset were not in an RGB format but were in an CMYK format. These images were converted to RGB format using python and the images have been imported as a .mat file. Due to restricted computational resources the images also have been reduced to 64 x 64 dimensions from 224 x 224 dimensions.

2.1 Class balancing

The initial dataset distribution was uneven, with certain classes having a much larger number of images compared to others. To mitigate this bias, I balanced the dataset through oversampling the under-represented classes and undersampling the over-represented ones. After balancing the classes all the classes consists 2617 images making it a total of 26170 images accross all the classes. The sample plot of an images in each class and the labels can be seen in figure 1.



(a) Sample images



Class	Animal
1	Dog
2	Horse
3	Elephant
4	Butterfly
5	Chicken
6	Cat
7	Cow
8	Sheep
9	Spider
10	Squirrel

(b) Labels

Figure 1: Sample Images and labels

2.2 Reshaping

I reshaped the original data which was 4-dimensional (number of images, height, width, color channels) into a 2-dimensional matrix. The image data is now of the size 26170 x 12288 (number of images x dimensions).

2.3 Normalization

After converting the data into a 2D matrix I have normalized the data and also converted the values into double. I have normalized each pixel value by dividing it by 255. This normalization helps machine learning algorithms process the data more efficiently.

2.4 Centering

To improve the model's training I centered and normalized by standard deviation. First, I centered all the data around the mean of the images getting the data distribution closer to zero. Then I normalized the data again using the standard deviation of the data. Normalization by standard deviation ensures that all features (pixels in image data) contribute equally to the distance calculations used by many machine learning models (like k-nearest neighbors or distance-based classifiers).

3 Splitting of data

I have split my data using a 80:10:10 ratio. The 80% training data provides ample material for your model to learn. The 10% validation and test sets, while smaller, still have enough images for effective hyperparameter tuning and unbiased evaluation without sacrificing a significant portion of training data. I split the data for raw data and ensured to use the same train, test, validation indices after reducing the dimensions using dimensionality reduction techniques.

4 Dimensionality Reduction Techniques

Dimensionality reduction refers to the process of transforming data from a high-dimensional space to a lower-dimensional space while aiming to preserve the most important information. As dimensionality increases, certain machine learning algorithms can suffer from the curse of dimensionality. Which refers to phenomenon like increased computational cost, data sparsity etc. These techniques address the curse of dimensionality by transforming the data into a lower-dimensional space with a smaller number of features. The techniques used in this project are

4.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical dimesionality reduction technique that simplifies high-dimensionality data by identifying the most informative direction (principal components) and projecting data onto a lower-dimensional space. It is a technique used for feature extraction which combines input variables in a specific way so that we can reject noise or less important data while still retaining the most valuable parts of information.

Working of PCA:

1. Standardization
   In this step we standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis. The data is centered around the mean by subtracting the mean value of each feature centering it around zero and then we divide it by standard deviation.

2. Covariance matrix computation
   PCA calculates the covariance matrix, which captures the linear relationships between each pair of features. A positive covariance indicates features move together, while negative suggests they oppose each other. Values near zero imply minimal connection.

3. Compute eigenvalues and eigenvectors
   Eigenvalue decomposition breaks down the covariance matrix, revealing the underlying structure of the data. Eigenvectors represent special directions within the data, and eigenvalues act as scaling factors associated with each eigenvector. They tell you how much variance (spread) each direction captures.

4. Identify principal components
   PCA generally sorts the eigenvectors based on their corresponding eigenvalues, with the highest one corresponding to most informative principal component (PC). This PC captures the direction of greatest variance in the data.

5. Dimensionality Reduction
   We do not retain all the components but only a certain number. This choice depends on the desired level of information preservation and data's characteristics. These commonly include explained variance ratio, scree plot.

6. Projecting the data
   Once we have chosen the principal components, we project the original data points onto the subspace spanned by these components. This creates a lower-dimensional representation of the data that captures the most significant variations.

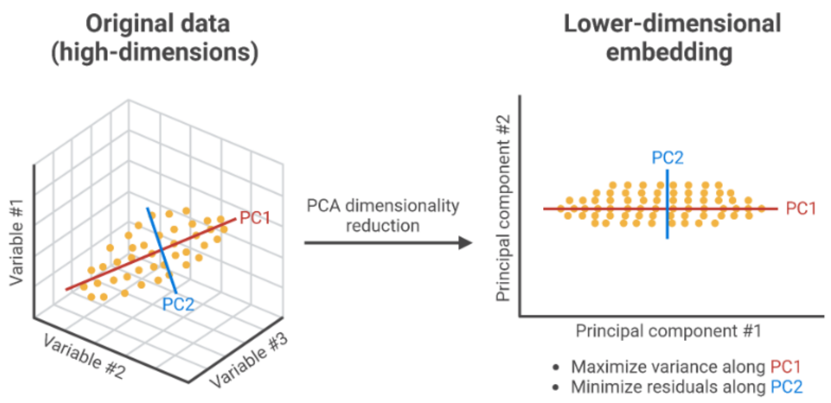

Original data (high-dimensions)
Variable #1
Variable #2
Variable #3
PC1
PC2
PCA dimensionality reduction
Lower-dimensional embedding
Principal component #2
Principal component #1
PC2
PC1
• Maximize variance along PC1
• Minimize residuals along PC2

Figure 2: PCA

4.2 Kernal Principal Component Analysis (KPCA)

Regular PCA struggles with non-linear data, focusing on linear relationships between features. Kernel PCA overcomes this limitation by employing a clever trick. It implicitly maps the data to a higher-dimensional space where these non-linear relationships become more linear. PCA is then performed in this new space, allowing it to capture the important variations even in non-linear data. In terms of computational complexity, PCA is generally faster and more computationally efficient than Kernel PCA.

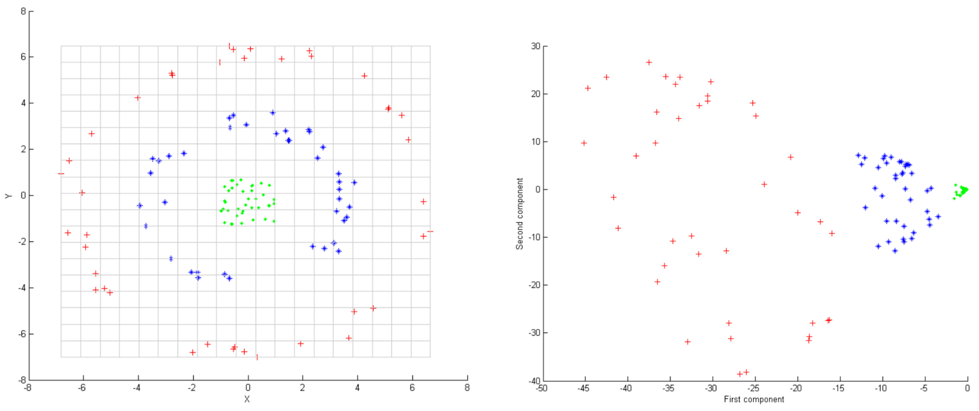
Working of Kernal PCA:

1. Kernel function
   Heart of KPCA lies in this function. The function takes two data points as inputs and calculates a similarity measure between them. It does this in a higher-dimensional feature space, even though we never explicitly define that space. Common kernel functions include linear, polynomial, and radial basis function (RBF).

2. Building kernel matrix
   By applying the kernel function to all pairs of data points, kernel PCA constructs a kernel matrix. This matrix essentially captures the pairwise similarities between data points in the high-dimensional feature space.

3. Eigenvalue decomposition
   Similar to PCA kernel PCA performs eigenvalue decomposition on the kernel matrix. However, the decomposition reveals the principal components within the high-dimensional feature space, where the non-linear relationships become more linear.

4. Dimensionality reduction
   Kernel PCA selects the principal components with the highest eigenvalues, representing the most variance in the high-dimensional space. These components capture the most significant non-linear relationships in the original data. Finally, kernel PCA projects the data points onto a lower-dimensional space using these principal components.



(a) Input Before KPCA

(b) Input After KPCA

Figure 3: KPCA

4.3 Locally Linear Embedding (LLE)

LLE, is a dimensionality reduction technique used for data visualization and manifold learning. Unlike PCA (Principal Component Analysis) which focuses on capturing variance in the entire data space, LLE aims to preserve the local structure of the data.

Working of LLE:

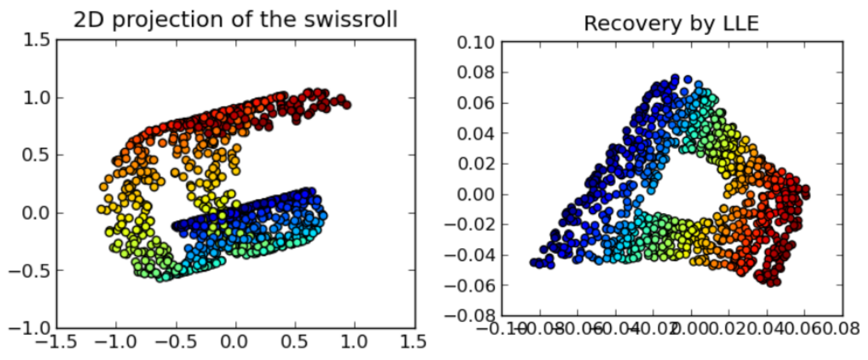
1. Define the neighborhood
   The first step involves defining a neighborhood for each data point. This neighborhood consists of its k nearest neighbors, essentially the data points closest to it in the

high-dimensional space. These neighbors represent the local structure surrounding each point.

2. Reconstructing each data point
   The core concept of LLE lies in reconstructing each data point using a linear combination of its k nearest neighbors. LLE finds weights for each neighbor, such that when you add them up proportionally, you get a reconstruction of the original data point. The key is to minimize the difference (reconstruction error) between the original point and its reconstruction.

3. Cost Function and Optimization
   To achieve the best reconstruction for all data points, LLE employs a cost function. This function essentially calculates the total reconstruction error for all data points combined. It searches for the best combination of neighbor contributions that accurately represent each point while preserving the local relationships.

4. Low-dimensional representation
   After the optimization process, we have the weights for reconstructing each data point from its neighbors. These weights, however, are in the high-dimensional space. We then project these weights onto a lower-dimensional space, often chosen to be Euclidean.



(a) Before LLE

(b) After LLE

Figure 4: LLE

5 K Nearest Neighbors

k-nearest neighbors algorithm (k-NN) is a supervised learning method and a simple classification algorithm used in machine learning. It is a non-parametric and instance-based learning algorithm, meaning it does not make any assumptions about the underlying data distribution and instead relies on the data instances themselves for classification. It is used for classification and regression. In both cases, the input consists of the k closest training examples in a data set.

Working of the algorithm

1. Training
   It does not involve any explicit training phase. Instead, it simply memorizes the training dataset, which consists of labelled data points in a feature space.
2. Classification When presented with a new, unlabeled data point, KNN classifies it based on the majority class of its nearest neighbors in the feature space.
   (a) Distance calculation
   KNN calculates the distance between the new data point and all points in the training dataset. Common distance metrics include Euclidean, Manhattan or Mikowski distance.
   (b) Neighbor selection:
   KNN selects the k nearest neighbors to the new data point based on the calculated distances. In our project we are considering the number of neighbors (k) to be 20. So, we will select 20 nearest neighbors to classify the new point. Once the k nearest neighbors are identified, KNN assigns the new data point to the class that is most common among its neighbors.
3. Prediction
   After determining the majority class among the k nearest neighbors, it assigns this class label to the new data point, thus predicting its class.

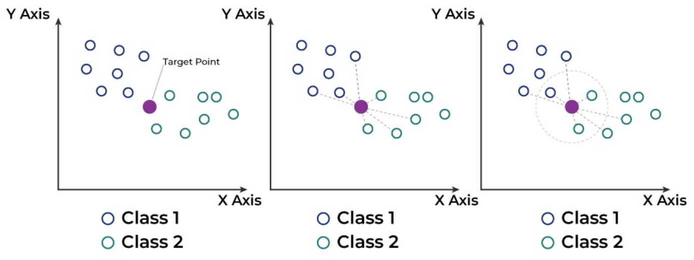


Figure 5: K Nearest Neighbors

6 Analysis

After deciding on the dimensional reduction techniques I want to use for the data I built a KNN classifier to compare the results of all the above techniques on the basis of 2 metrics

accuracy and F1 score.
I have kept the number of neighbors to be 20 for the KNN classifier constant for all the methods.

6.1 Raw data

I built a KNN classifier on the raw data after pre processing the data. I plotted a 2D plot of all the classes for comparison with other plots after reducing the dimensions. The plot can be seen below in figure 6.

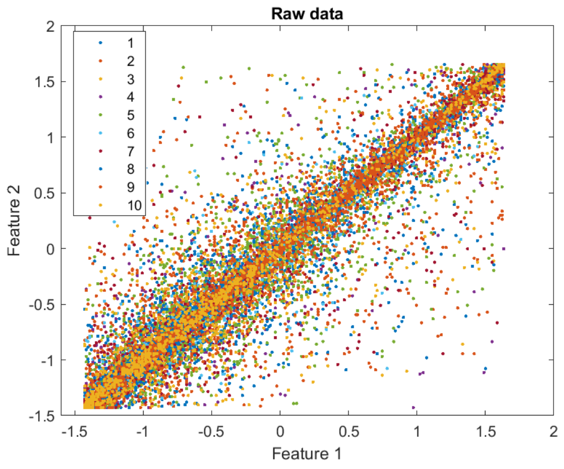


Figure 6: 2D plot of raw data

Metric	Test	Validation
Accuracy	25.98%	24.38%
F1 score	0.09	0.11

Table 1: Performance Metrics on Raw data

6.2 PCA

As mentioned above after performing PCA we only retain a certain amount of components and not all of them. The default when nothing is mentioned is to preserve 2 components. Due to limited amount of computational resources I was able to run PCA for only 6000 components which is almost half of the total components (64*64*3 = 12288). After running PCA to decide on the number of components to retain to transform the data I calculated explained varaince and plotted a screeplot which can be seen in figure 8. We can also see the 2D representation of the data after reducing it using PCA in figure 7.

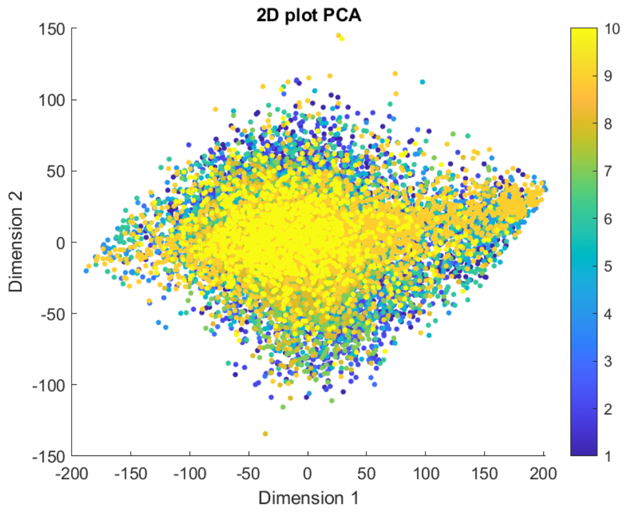


Figure 7: 2D plot of PCA data

In the plot mentioned above in figure 7, we can observe that the data points are more spread out and separated compared to the raw data representation. This is because PCA aims to find a lower-dimensional subspace that best preserves the variance in the data, potentially making it easier to separate the classes. However, some overlap between classes is still evident, suggesting that linear methods like PCA may not be sufficient for this particular dataset.

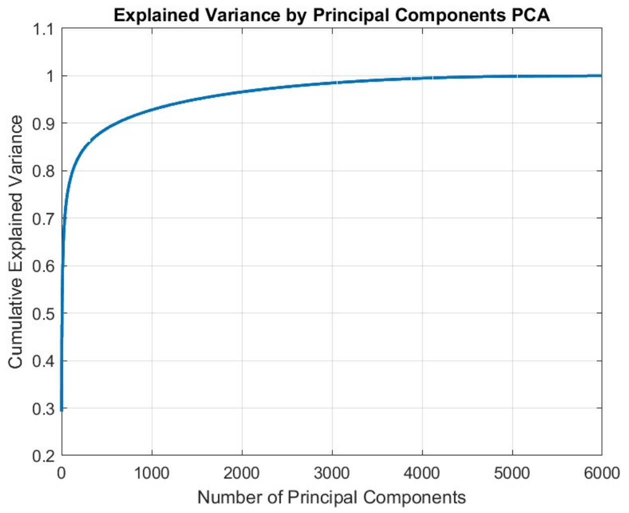
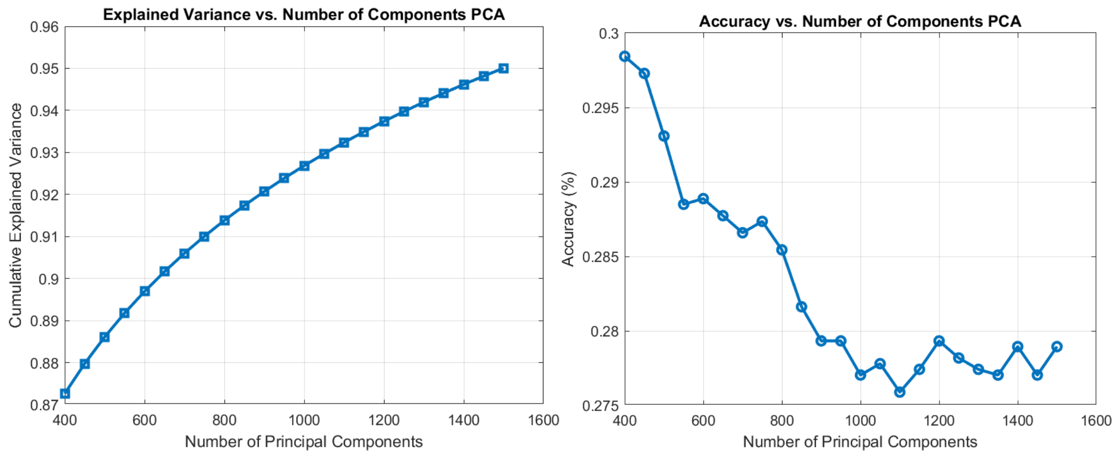


Figure 8: Scree plot of PCA components

I have considered a range of values in the elbow of the scree plot above and calculated the accuracy for all the values in increments of 50 and plotted graphs to pick the number of principal components to retain. The plots can be seen below in figure 9.



(a) Variance vs components (b) Accuracy vs components

Figure 9: PCA elbow components

From the figure 9 we can see that retaining 400 components has the highest accuracy which is 29.4% and it retains 87.2% of variance where as 650 components are giving an accuracy of 28.8% but are retaining a variance of 89.6%. Striking a balance between the variance retained as well as getting a good accuracy is important so I have considered to retain 600 components and the metrics can be seen in the below table 2.

Metric	Test	Validation
Accuracy	28.89%	27.32%
F1 score	0.12	0.10

Table 2: Performance Metrics on PCA

6.2.1 PCA on complete data vs split data

I have performed PCA on the complete data and then split it into train, test and validation sets as well as separately after splitting them to see how the classifier will behave. The results of both the analysis can be seen in the table 3 and 4 below.

Metric	Test	Validation
Accuracy	24%	25.3%
F1 score	0.07	0.09

Table 3: Performance Metrics on PCA performed **AFTER** splitting

Metric	Test	Validation
Accuracy	29%	28%
F1 score	0.12	0.10

Table 4: Performance Metrics on PCA performed **BEFORE** splitting

We can observe that performing PCA on the dataset before splitting has turned out to be beneficial in our case. I think the below mentioned might be a few reasons of why I might be getting better accuracy with the total dataset approach which are more information retained, consistent transformations, more data for transformation, preservation of global structure.
We can observe that the accuracy of our model improved when compared to raw data. PCA works well when the relationships between features are linear. However, in image data, especially animal images, the relationships between pixel values can be highly nonlinear due to variations in color, texture, shape, and other visual attributes. Which is why we can see the accuracy of our model did not improve by a lot.

6.3 Kernel PCA

I was unable to perform KPCA to preserve a said number of components like PCA due to computational resources. After reducing the number of components from 6000 till 100 I was always encountering the error of 'out of memory' hence had to leave the number of components to default (i.e, 2). the 2D representation of the data can be seen in the figure 10 below.

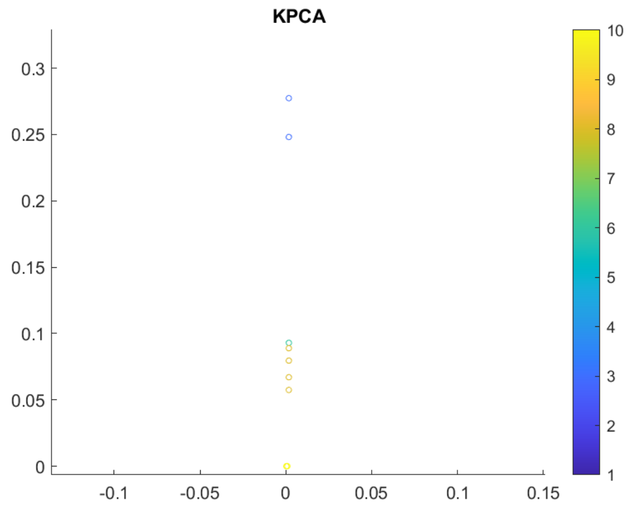


Figure 10: 2D representation of KPCA

Due to 'out of memory' error I was unable to generate a proper plot of the 2D representation of the KPCA transformed data.

Metric	Test	Validation
Accuracy	36.22%	37.29%
F1 score	0.28	0.28

Table 5: Performance Metrics on KPCA

We know that PCA finds linear combinations of the original features KPCA, on the other hand, is well-suited for capturing nonlinear relationships in image data by leveraging kernel functions. It can map data onto higher dimensional feature space where non-liner relationships are easier to capture. This allows KPCA to extract more meaningful features from the image data, capturing complex patterns and variations that PCA might miss. Animal images often contain complex patterns and variations that can be well captured using certain non-linear techniques using KPCA making it a better fit and more effective

6.4 LLE

To reduce the dimensions of my data using LLE I have changed the value of the the number of neighbors that LLE will consider from 2 to 30 to see how the accuracy of the classification model will change with the change in the number of neighbors. Few of the results are discusses

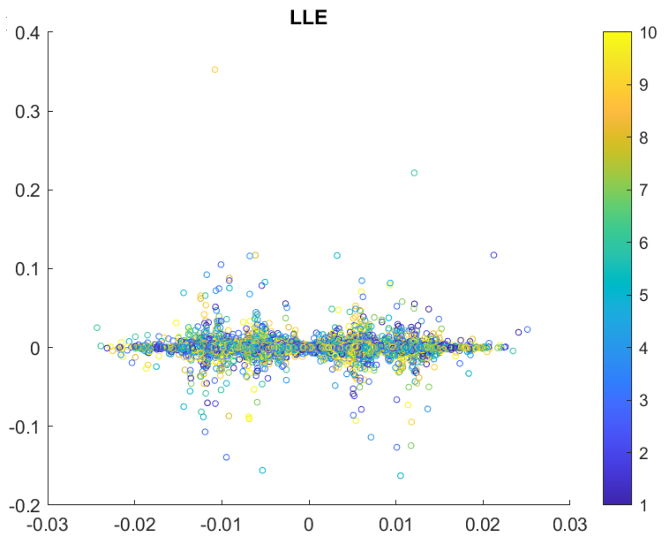


Figure 11: 2D representation of LLE

In the plot (figure 11) above, we can observe that the data points are more clustered and separated compared to both the raw data and PCA representations. While there is still some overlap between classes, LLE appears to have done a better job of separating the classes, especially for certain clusters or groups within each class.

Number of neighbors	Accuracy
5	16.29%
8	16.71%
10	17.33%
12	14.97%
20	15.44%
30	17.41%

Table 6: Performance Metrics on LLE with different neighbors

We can see above that the best accuracy for LLE occurred when we chose 30 neighbors which is closely followed by 10 neighbors. The number of neighbors in locally linear embedding (LLE) really matters. If we choose too many neighbors, like 30, LLE might smooth out the important details in the data and get confused by noise, which could make the results less accurate. Also, it might take longer to run and it does use up more computer resources. On the other hand, choosing fewer neighbors, like 10, strikes a better balance. It focuses on the most important details in your data while still running efficiently.

Locally Linear Embedding (LLE) aims to preserve the local relationships between data points which does not seem to be a good fit for the data we are using. LLE is usually effective for data that contains clear local structure which might not be seen in the animal images of our data due it's low accuracy. It may not be performing well with image data due to its high dimensionality and complex relationships between pixels. The lower accuracy with LLE could be attributed to its difficulty in capturing the intricate patterns and variations present in animal images.

6.5 Isomap

Usually ISO map is considered a good fit for image data to reduce the number of dimensions. I tried reducing the dimensions using Isomap but after running for almost 12 hours I encountered an error of 'out of memory'.

7 Comparision

The below table 7 contains the values of 2 metrics of all the above methods.

Method	Accuracy	F1 score
Raw data	25.98%	0.09
PCA	29.89%	0.12
KPCA	36.22%	0.28
LLE	17.33%	0.16

Table 7: Performance Metrics on all methods

KPCA performed the best in terms of both accuracy and F1 score, likely due to its ability to capture non-linear patterns in the image data through kernel functions.

PCA, a linear dimensionality reduction technique, improved accuracy over raw data by removing redundant or irrelevant features, but its linear nature may have limited its performance compared to non-linear methods, as reflected in its lower F1 score than LLE.

LLE, another non-linear method that focuses on preserving local structures, had a higher F1 score than linear methods, suggesting it could better capture local patterns crucial for distinguishing between animal classes, although its lower accuracy indicates challenges in generalization.

The raw data, without any dimensionality reduction or transformation, performed the worst, as the high-dimensional and potentially noisy image features made it difficult for machine learning algorithms to effectively learn the underlying patterns.

I would explore more in detail on which classes were classifying the worst to see why they are behaving a certain way. I was unable to compute that for even the dimensional reduction technique with gave me the highest accuracy i.e, KPCA as my matlab went out of memory and I could not run KPCA any longer.

8 Conclusion

This project compared various data representation methods - raw data, PCA, kPCA, and LLE - for classifying 10 animal types from image data. The nonlinear kPCA method performed best, achieving 36.22% accuracy, likely due to its ability to map data into higher dimensions and capture complex patterns through kernel functions. While PCA improved over raw data by reducing dimensionality, its linear nature limited performance compared to nonlinear approaches like kPCA. LLE demonstrated potential by preserving local structures (higher F1 score) but struggled with generalization (lower accuracy).

Overall, the results highlight the importance of considering nonlinear methods for image classification tasks, as they can better model underlying patterns compared to linear techniques or raw representations.