CIFAR-10 dataset
KNN classification - Exploring data representation methods

Name: Swathi Baskaran

Objective:

This project investigates how different ways of representing data affect image classification on the CIFAR-10 dataset. We will compare four methods: using the raw images directly, Principal Component Analysis (PCA), kernel PCA (kPCA), Locally Linear Embedding (LLE). For each representation, I will train a K-Nearest Neighbors (KNN) classifier and measure its performance in terms of classification accuracy and computational efficiency (training time). The main goal is to understand how these representation methods influence the KNN classifier's ability to accurately classify images, while also considering the computational cost.

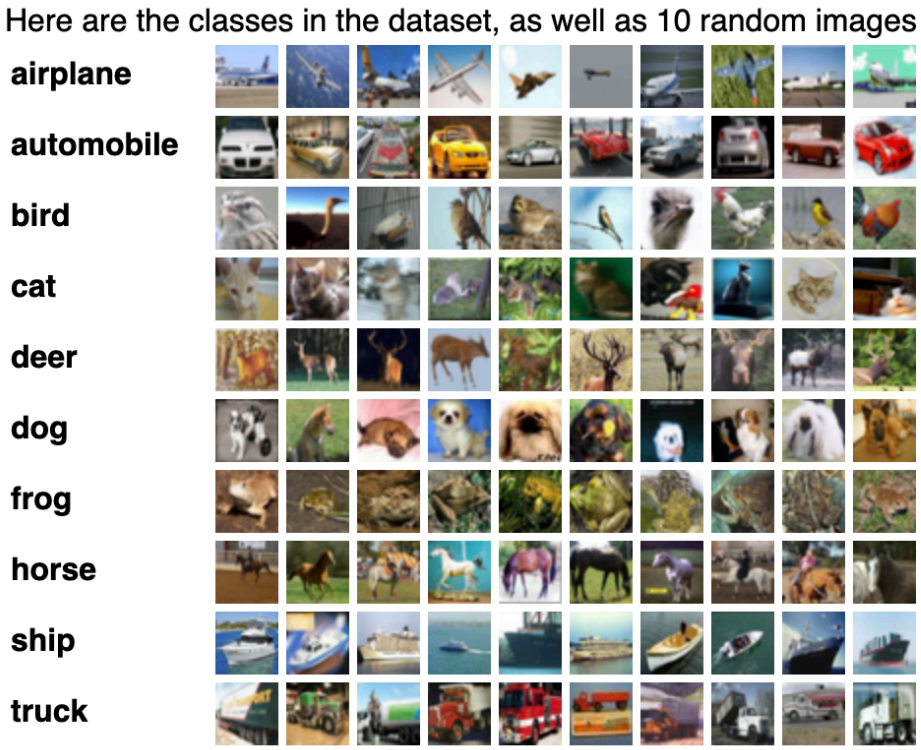
Dataset Description:

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. The originally provided dataset included a separate training set with 50,000 images and test set with 10,000 images. Interestingly, the training set was further divided into five batches of 10,000 images each and the test batch contains exactly 1000 randomly-selected images from each class. To improve the training process, I combined all the batches into a single dataset of 60,000 images. I normalized and reshaped the data after combining it, so dimension of the data is now 3072 (32x32x3). This larger dataset was then split into training, validation, and test sets for the classification task.

Name: CIFAR-10 (Canadian Institute for Advanced Research)
Dataset Link: https://www.cs.toronto.edu/~kriz/cifar.html
Size: 60,000 images (50,000 training + 10,000 testing)
Image format: 32x32 pixels, RGB color

Number of classes: 10 (6000 images per class)

Class Labels: The CIFAR-10 dataset consists of images belonging to 10 mutually exclusive classes, representing everyday objects.



Data splitting:

To evaluate the impact of data splitting strategies on KNN classification performance, I experimented with few data splitting methods. Then finalized 2 prominent methods, random sampling and stratified sampling after exploring various methods. For each method, I explored various train-test-validation set splits, including 60-20-20, 70-15-15, 70-10-20, and 80-10-10. I then trained KNN classifiers on each resulting dataset and recorded their corresponding accuracies. To finalize on the splitting method and ratio, I spent 3-4 hours to learn and experiment.

The following table summarizes the results:

Split ration/method	Training set	Validation set	Test set	Validation Accuracy	Test Accuracy
Random Sampling					
60-20-20	36,000	12,000	12,000	31.64%	31.95%
70-15-15	42,000	9,000	9,000	32.46%	31.77%
70-10-20	42,000	6,000	12,000	32.34%	31.68%
80-10-10	48,000	6,000	6,000	32.26%	32.33%
Stratified Sampling					
60-20-20	36,000	12,000	12,000	32.84%	32.13%
70-15-15	42,000	9,000	9,000	32.72%	32.85%
70-10-20	42,000	6,000	12,000	32.58%	32.89%
80-10-10	48,000	6,000	6,000	33.63%	32.71%

Method: I decided to go with stratified sampling because it gave better accuracies and it fit my dataset well. Stratified sampling is a method used to ensure that the distribution of classes (or labels) in the dataset remains consistent across the different subsets (train, validation, and test sets). By preserving the class distribution, models trained on stratified data splits are

- More likely to generalize well to unseen data
- Reduces variability in model performance
- Prevents biases in performance assessment

This is particularly useful when dealing with imbalance in dataset where certain classes may be underrepresented. Ultimately, I decided to use stratified data splitting to get a more reliable model evaluation and achieve better performance in classification tasks.

Split ratio: After experimenting on different data split ratios like 60-20-20, 70-15-15, 70-10-20 and 80-10-10 for train-validation-test ratio respectively, I found that the 80-10-10 ratio resulted in the highest accuracy. However, upon further examination, I realized that the dataset required more training data, particularly as the images were only 32x32 pixels and lacked image clarity. So, the model might need more data to train on so decided to use the 80-10-10 split.

KNN classification methods with different representation methods:

1. Raw Data:

I performed KNN classification on the raw data of the CIFAR-10 dataset. I decided to use n_neighbors=20 constant for KNN classification throughout. The validation accuracy was 33.63% with a corresponding F1 score of 0.32. The test set performance yielded a slightly lower accuracy of 32.72% and an F1 score of 0.31. The code ran in **17.26secs**. These results suggest that the KNN model struggles to learn effective classification boundaries from the raw pixel data. Moreover, I examined the class-wise accuracy to gain a deeper understanding of our model's performance across different categories.

- The model achieved the highest accuracy for the 'ship' class, with 71.83% on the validation set and 72.00% on the test set, indicating its proficiency in distinguishing ships from other objects.
- The 'automobile' and 'truck' classes exhibited the lowest accuracies, underscoring the challenges in accurately classifying these categories.

This suggests that the KNN model might be better at classifying objects with distinct visual features (like ships) compared to those with more subtle variations (like cars). It's important to note that these results will be used as a baseline for comparison as we explore the impact of dimensionality reduction techniques like PCA and kPCA on KNN classification performance.

Here is the table with the recorded accuracy:

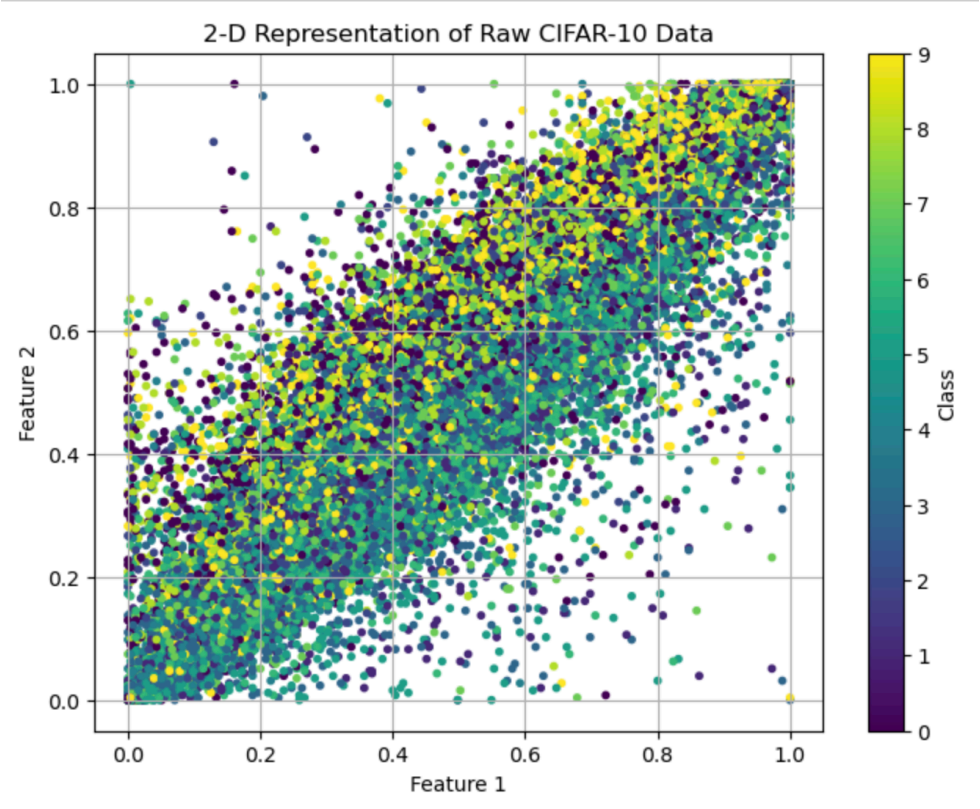
	Accuracy	F1-score
Validation Set	33.63%	0.32
Test set	32.72%	0.31

Class	Validation Accuracy	Test Accuracy
airplane	49.67%	49.83%
automobile	13.00%	13.67%
bird	43.33%	39.17%

Class	Validation Accuracy	Test Accuracy
cat	15.83%	14.17%
deer	58.83%	53.17%
dog	23.67%	23.00%
frog	28.17%	27.50%
horse	17.05%	18.83%
ship	71.83%	72.00%
truck	14.50%	15.83%

This is the 2-D representation of raw data:

Each data point on the plot represents a single image from the CIFAR-10 dataset. This plot provides a glimpse into a preprocessed version of the CIFAR-10 dataset. It helps visualize how different image features might be distributed and potentially reveal groupings based on object class.



2. PCA

Principal component analysis (PCA) is a dimensionality reduction technique that can be used to reduce the number of features in a dataset. It does this by finding a new set of features, that are linear combinations of the original features.

In the context of the CIFAR-10 dataset, PCA prioritizes features that explain the most prominent variations in the images. These variations would include aspects such as the overall color distribution, the presence of distinct patterns or textures, and the shapes of the objects depicted in the images. By extracting principal components, PCA aims to capture the distinguishing characteristics across different classes, and help us tell apart airplanes, cars, birds, cats, and other things in the pictures.

Here's how PCA works:

Finding Major Axes of Variation: PCA analyzes the high-dimensional data and identifies the directions (principal components) that exhibit the most significant variance. These directions represent the underlying factors that differentiate the data points in the dataset.

Projecting Data onto PC: Imagine the high-dimensional data as a cloud of points in a complex space. PCA identifies a new coordinate system defined by the principal components. It then projects the data points onto this new coordinate system, essentially transforming them into a lower-dimensional space while preserving the maximum variance.

Prioritizing Informative Components: During this projection, PCA prioritizes the components that capture the most variance. These components represent the most informative features that distinguish between different classes in the data. Components with less variance are considered less important and might be discarded, depending on the desired level of dimensionality reduction. With fewer dimensions after PCA, there's a lower risk of overfitting the model to the training data. This can lead to better generalization performance on unseen data, resulting in improved accuracy.

Optimal (n_component) -> Balancing Variance and Dimensionality with PCA for CIFAR-10 Classification:

When applying PCA, it's essential to choose the number of principal components (n_components) that maximizes the information retained while minimizing the dimensionality effectively. While doing this, a crucial trade-off exists between preserving informative variance and reducing dimensionality. I experimented this trade-off by analyzing the cumulative explained variance ratio versus the number of components (0-3072).

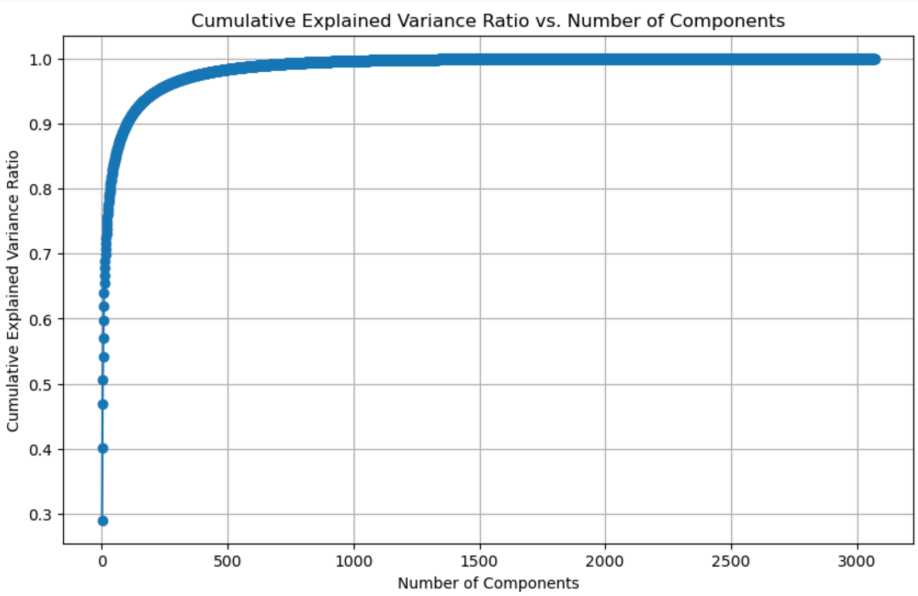
I began by experimenting with different numbers of components. I started with 1536 (half of the original 3072), then halved that number to 768, and continued down to 200, 100, 50, and even 20 components. After this trial-and-error process, it was hard to find an optimal number of components. While reducing the number of components generally led to improved accuracy, using too few components will not help in training the model. Through this I understood the trade-off between accuracy and dimensionality. To understand this better, I decided to do the plots for the components. Here is the table for the accuracies for few of the trailed experiments.

Number of Components	Validation set	Test set
1536	33.67%	32.73%
500	34.12%	33.13%
200	35.62%	34.95%
50	39.52%	39.27%
20	41.42%	40.51%

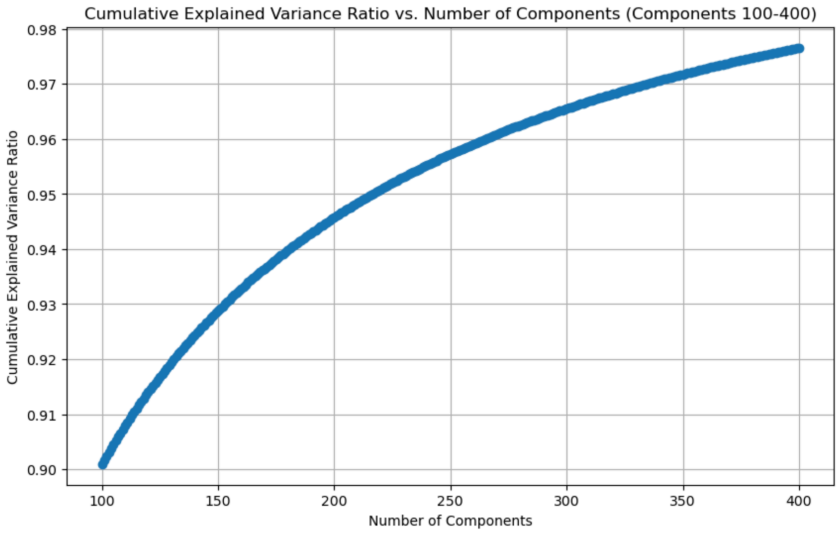
The analysis revealed that using a very high number of components (>600) retains a significant portion of the variance, but it also leads to a high-dimensional space, increasing computational cost and potentially hindering KNN performance. With very high dimensionality (all 3072 components), KNN can suffer from the "curse of dimensionality." This means as the number of features increases, training machine learning models can become computationally expensive and less effective, making KNN less effective.

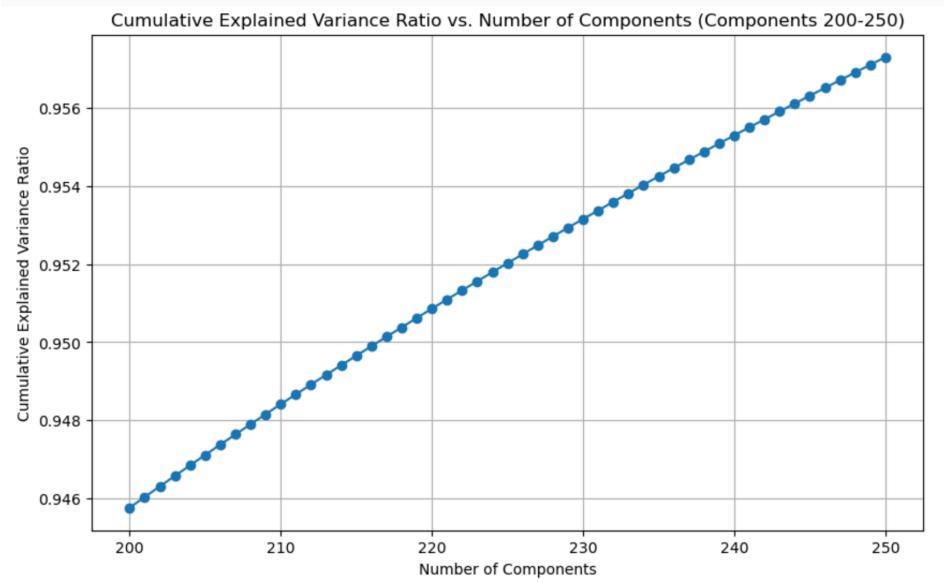
Conversely, using too few components (<200) offers lower dimensionality, potentially improving computational efficiency, but might discard crucial information, limiting KNN's ability to learn

effective classification boundaries. Accuracy may decrease if you choose a value of n_components that is too low or too high for your dataset, leading to either underfitting or overfitting.

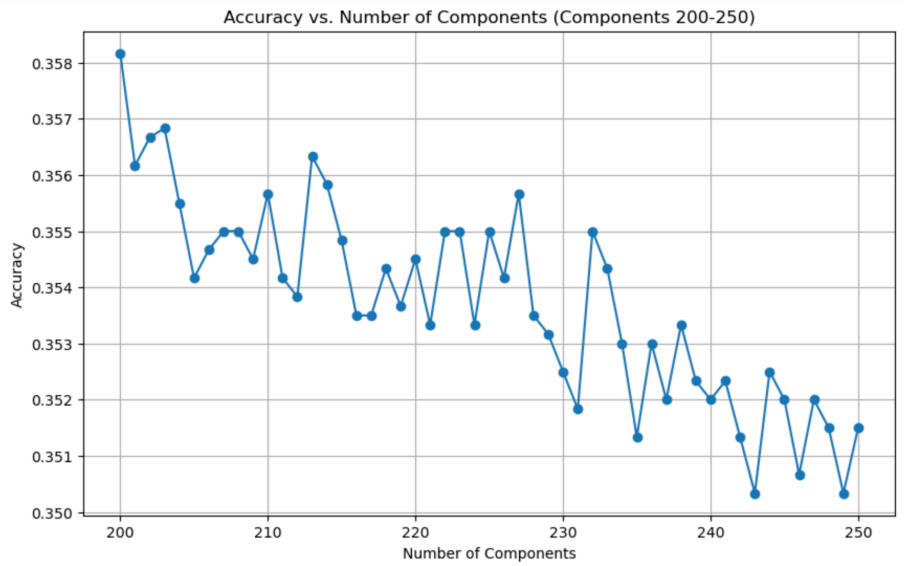


After studying the plot, I focused on the range between 100 and 400 components, to achieve a balance. I wanted to capturing most informative features, minimizing information loss and avoiding overfitting so I was aiming for at least 95% of the variance preserved because the images lacked clarity. After more granular analysis, I plotted a graph between 200 and 250 components.





I also monitored the KNN model's accuracy for different component values.



After studying and doing combined analysis of both the plots, I identified 227 components as a suitable choice. This selection retains a high percentage of variance (over 95%) while offering a reasonable reduction in dimensionality. Additionally, the KNN model achieved a promising accuracy using this number of components compared to models trained with fewer or more components.

KNN classification:

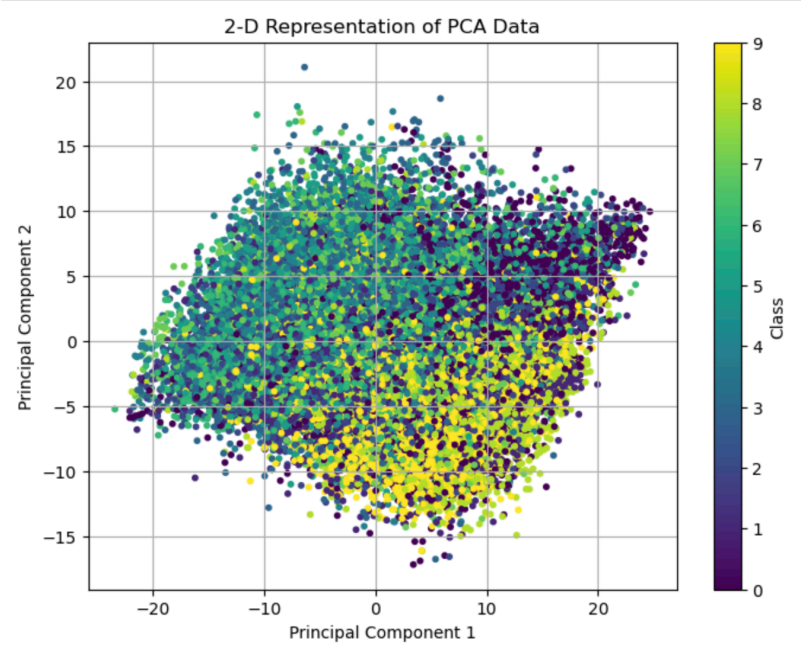
Applying PCA for dimensionality reduction demonstrably improved the performance of KNN classification. Compared to using raw data, PCA yielded a slight increase in validation accuracy (35.48% vs 33.63%) and almost same F1 score (0.31 vs 0.32). While the test set results showed a similar trend (accuracy: 34.33% vs 32.72%, F1 score: 0.33 vs 0.31), the most significant improvement is observed in the class-wise accuracy. The PCA code ran in **14.12 seconds.** I ran the code for 30-40 times. Computational cost for PCA: O(D^3) where D is the number of dimensions.

Several classes, particularly "automobile" and "truck", exhibited a noticeable improvement in validation accuracy (17.50% vs 13.00% and 18.00% vs 14.50% respectively). This suggests that PCA effectively captured underlying patterns in the data that aided KNN in differentiating these classes. Even for classes like "ship" that maintained a high accuracy with raw data (71.83%), PCA preserved this performance (71.33%). Here are the tables with the recorded accuracy:

	Accuracy	F1-score
Validation Set	35.48%	0.31
Test set	34.30%	0.33

Class	Validation Accuracy	Test Accuracy
airplane	50.83%	51.67%
automobile	17.50%	18.00%
bird	43.00%	38.67%
cat	15.00%	11.67%
deer	55.67%	49.67%
dog	23.00%	21.17%
frog	38.00%	38.67%
horse	23.00%	21.83%
ship	71.33%	71.67%
truck	17.50%	20.00%

Shown below is the 2-D representation of the PCA transformed data:



3. kPCA

While PCA excels at capturing linear variations in data, kernel PCA (kPCA) offers a powerful alternative for datasets exhibiting non-linear relationships between features. In the context of the CIFAR-10 dataset, kPCA goes beyond identifying features like overall color distribution or basic shapes. It can potentially uncover more complex patterns that might be crucial for differentiating certain image classes.

Here's how kPCA works:

Non-Linear Mapping: kPCA first projects the data points into a higher-dimensional feature space using a kernel function. This allows kPCA to capture non-linear relationships that wouldn't be readily apparent in the original data space. With fewer dimensions, the model focuses on the most relevant information, potentially leading to better generalization on unseen data.

Dimensionality Reduction: Similar to PCA, kPCA then identifies the principal components within this high-dimensional space. These components represent the most significant variations in the data after the non-linear transformation.

Reduced Dimensionality Space: Finally, kPCA projects the data points back into a lower-dimensional space, retaining the most informative components from the high-dimensional representation. This compressed space offers benefits similar to PCA, such as reducing computational cost and potentially improving KNN performance.

Handling Non-Linearities: kPCA's ability to capture non-linear relationships can be advantageous for KNN in CIFAR-10. Certain classes, like different types of animals, might have subtle variations in their visual features that PCA might miss. kPCA's ability to model these non-linearities can potentially improve KNN's classification accuracy for such classes.

However, it's important to consider computational cost. kPCA can be computationally more expensive compared to PCA, especially for large datasets like CIFAR-10. Similar to PCA, I used 227 as the number components for kPCA as well. The effectiveness of kPCA relies heavily on choosing the right kernel function. I considered 2 different kernels - Radial Basis Function (RBF) and Polynomial Kernel. I chose accuracy over computational complexity and chose 'rbf' kernel over 'poly' kernel. The code ran in **1878. 35 seconds**, that is almost 30-40 mins for each run. The computational complexity of computing the kPCA is usually O(N^2 D), where D is the number of dimensions.

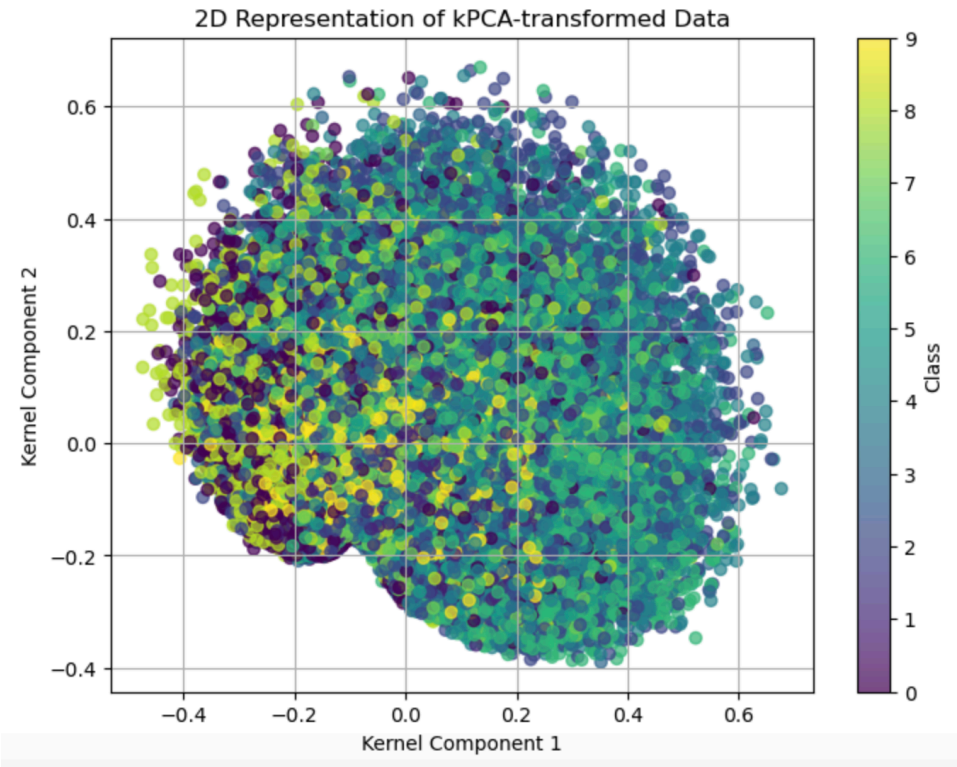
RBF Kernel	Computationally costly	Better Accuracy
Poly Kernel	Computationally less costly	Slightly lesser accuracy

kPCA significantly improved model performance, achieving a 10 percentage point increase in both validation and test accuracy compared to using raw data. There is also substantial rise in accuracy for many individual classes, the 'automobile' class saw a significant increase from 13.00% (raw data) to 61.00% (kPCA) on the validation set, and from 13.67% to 63.50% on the test set. Similarly, improvements were observed in other classes such as 'ship', 'truck', and 'horse'. This suggests that kPCA, by leveraging kernel tricks to map the data into a higher-dimensional space, effectively captures nonlinear relationships in the CIFAR-10 dataset, leading to improved discrimination between different classes and overall enhanced classification performance. Here is the table with the recorded accuracy:

	Accuracy	F1-score
Validation Set	43.60%	0.43
Test set	44.63%	0.44

Class	Validation Accuracy	Test Accuracy
airplane	50.00%	55.83%
automobile	61.00%	63.50%
bird	28.67%	26.83%
cat	25.50%	24.17%
deer	29.5%	28.33%
dog	35.00%	36.17%
frog	49.33%	54.00%
horse	47.67%	47.33%
ship	51.00%	49.17%
truck	58.33%	61.00%

Here is the 2-D representation ok kPCA transformed data:



4. LLE

Locally Linear Embedding (LLE) offers another approach to dimensionality reduction for KNN classification in the CIFAR-10 dataset. Unlike PCA and kPCA, which focus on capturing global variance or non-linear relationships, LLE aims to preserve the local structure of the data, by representing each data point as a linear combination of its nearest neighbors. It then embeds the data into a lower-dimensional space where these local relationships are preserved as much as possible. This capability enables LLE to accurately represent intricate structures, even in datasets with complex patterns like CIFAR-10. LLE excels at capturing the underlying geometry of the data manifold and can adeptly handle nonlinearities. Additionally, its computational efficiency ensures swift computation of lower-dimensional representations, vital for managing large datasets such as CIFAR-10.

Unlike PCA and kPCA, which prioritize global patterns or non-linear trends, LLE excels at preserving the local structure of the data. This can be particularly beneficial for KNN in this dataset. Imagine images of different dog breeds. In the high-dimensional space, these images might reside close together despite variations, reflecting their underlying similarity. LLE's ability to maintain these local relationships can significantly improve KNN's accuracy in classifying such cases. While not explicitly designed for non-linearities, LLE can implicitly capture some localized non-linear variations within these neighborhoods. LLE provides a unique way to reduce dimensionality by focusing on the data's local geometry. This focus on local structure has the potential to significantly enhance KNN classification in CIFAR-10 by preserving both class similarities and implicitly capturing localized non-linearities.

Here's how LLE works:

Nearest Neighbors: LLE identifies the k-nearest neighbors for each data point in the high-dimensional space.

Local Reconstruction: It then reconstructs each data point using a linear combination of its neighbors. This process essentially captures the local geometry of the data manifold.

Dimensionality Reduction: Finally, LLE projects the data points into a lower-dimensional space while trying to maintain the reconstructed local structure from the high-dimensional space.

It is important to considered the optimal k neighbors for the data. So I tried experimenting on different k neighbor values for LLE. Here is the table with recorded values:

K neighbor	Validation accuracy	Test Accuracy	Computational Time
10	34.23%	35.02%	3524.81 sec
15	34.67%	34.45%	3511.07 sec
20	35.80%	35.03%	3393.10 sec

Running this was computationally expensive, so I was only able to experiment on 3 different ways. I didn't try higher values because the very high accuracy might indicate overfitting. I chose k=20 as it yielded a higher accuracy, it makes sense because it gives us a more detailed view of how each image relates to its neighbors. With k=20, we're considering more nearby images when creating the embedding, which helps us understand the structure of the dataset better. k=20 setting strikes a good balance between capturing the details of the images and making sure the model doesn't get too focused on small differences.

The code ran in **3393.10 seconds**, which nearly 1 hour for each run. I ran the code 4-5 times. The computational cost for LLE is O[D log(K) N log(N)] + O[D N K^3] + O[d N^2] where, N = number of data points, K = number of nearest neighbors, D = input dimension, and d = output dimension.

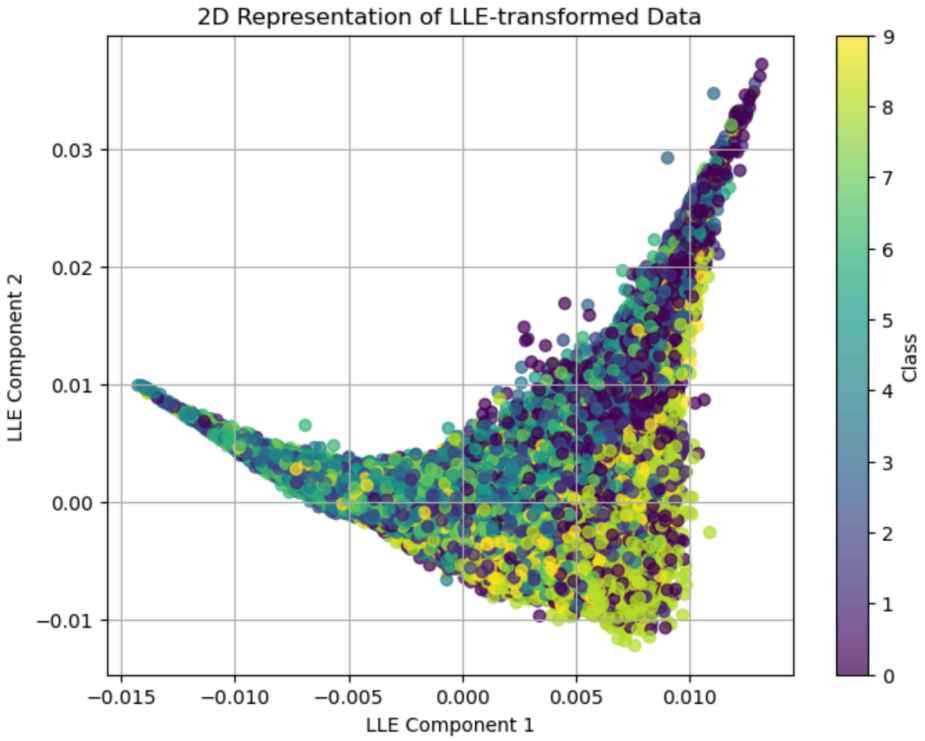
When comparing the performance of LLE with raw data representation, LLE achieved a slight increase in validation accuracy (35.80% vs 33.63%) and F1 score (0.36 vs 0.32), and in test accuracy(35.03% vs 32.72%) and F1 score (0.35 vs 0.31). However, in class-wise accuracy, there is varied improvements across different classes with LLE. Notably, classes like 'airplane', 'automobile', and 'ship' exhibited noticeable increases in accuracy with LLE, indicating its effectiveness in capturing local relationships and enhancing discrimination between these classes. Conversely, some classes such as 'bird', 'cat', and 'deer' showed limited improvements, suggesting that LLE may not effectively capture the inherent structure of these classes as

compared to others. Overall, while LLE presents modest improvements over raw data representation. Here is the table with the recorded accuracy:

	Accuracy	F1-score
Validation Set	35.80%	0.36
Test set	35.03%	0.35

Class	Validation Accuracy	Test Accuracy
airplane	43.17%	44.33%
automobile	40.67%	41.67%
bird	21.83%	22.17%
cat	20.00%	20.17%
deer	26.33%	23.17%
dog	32.33%	29.83%
frog	47.33%	45.83%
horse	36.00%	35.67%
ship	46.33%	41.67%
truck	44.00%	45.83%

Here is the 2-D representation of LLE transformed data:



Comparative Analysis:

Here's a tabular column summarizing the comparative analysis of the different representation methods for KNN classification on the CIFAR-10 dataset:

Technique	Description	Focus	Benefits for KNN	Drawbacks
Raw Data	No dimensionality reduction applied	-	Baseline for comparison	- High dimensionality can lead to high computational cost and potential overfitting.
PCA	Captures the most prominent variations in the data	Global data variance	- Reduced noise and redundancy - Improved efficiency - Focuses on informative features (e.g., color distribution, textures, shapes)	- May not capture non-linear relationships between features.
kPCA	Applies PCA after a non-linear transformation of the data.	Non-linear relationships	- Can capture complex patterns missed by PCA - May improve classification of classes with subtle variations	- Higher computational cost compared to PCA - Choosing the right kernel function is crucial.
LLE	Preserves the local structure of the data	Local data geometry	- Can improve classification for classes with similar local features (e.g., different dog breeds) - May implicitly capture some non-linear relationships	- Less emphasis on global patterns - Choosing the number of neighbors (k) is important.

Certainly, here's a tabular column summarizing the comparative analysis with numerical values:

Technique	Validation accuracy	Test Accuracy	Validation F1-score	Test F1-score	Computational cost
Raw Data	33.63%	32.72%	0.32	0.31	Low
PCA (227 components)	35.48%	34.22%	0.31	0.33	Moderate (O(D^3))

Technique	Validation accuracy	Test Accuracy	Validation F1-score	Test F1-score	Computational cost
kPCA (RBF Kernel, 227 components)	43.60%	44.63%	0.43	0.44	High (O(N^2 D))
LLE (K = 20 neighbors)	35.80%	35.03%	0.36	0.35	High (O[D log(K) N log(N)] + O[D N K^3] + O[d N^2])

Here is the advantage and disadvantage for each method:

Technique	Advantage	Disadvantage
Raw Data	Preserves all information in the images	High dimensionality (3072 features) challenging for KNN
PCA	Captures linear variations, modest performance improvement	Limited ability to handle non-linear relationships
kPCA	Captures non-linear relationships, significant performance improvement	Computationally expensive, especially for large datasets
LLE	Preserves local structure, modest performance improvement	Varying effectiveness across different classes, limited ability to capture global non-linearities

In these tables mentioned above, the key metrics compared are validation accuracy, test accuracy, validation F1-score, test F1-score, computational cost, advantages, and disadvantages of each representation method. The kPCA method stands out with the highest validation and test accuracies, but also the highest computational cost. PCA offers a balanced trade-off between performance and computational efficiency, while LLE provides modest improvements over raw data with varying effectiveness across classes.

Conclusion:

Among the representation methods evaluated, kPCA emerged as the best choice, outperforming raw data, PCA, and LLE in terms of both validation and test accuracy, F1 scores, as well as class-wise performance.

Specifically, kPCA achieved a validation accuracy of 43.60% and a test accuracy of 44.63%, along with validation and test F1 scores of 0.43 and 0.44, respectively. These results represent a

significant improvement over the baseline raw data representation and the other dimensionality reduction techniques.

The key advantage of kPCA lies in its ability to capture non-linear relationships within the data, which is particularly beneficial for complex image datasets like CIFAR-10. By leveraging kernel tricks to map the data into a higher-dimensional space, kPCA can effectively model the intricate patterns and variations present in the images, leading to improved discrimination between different classes. Moreover, kPCA demonstrated substantial improvements in the classification accuracy for several challenging classes, such as 'automobile' and 'truck,' further highlighting its effectiveness in handling complex visual features.

While kPCA comes with a higher computational cost compared to PCA and LLE, its superior performance justifies the additional computational overhead, especially if computational resources are available. Therefore, considering the significant performance gains, the ability to handle non-linear relationships, and the overall improvement in classification accuracy across various classes, kPCA emerges as the best representation method for the KNN classification task on the CIFAR-10 dataset.

———————— x x x ————————