Kernel PCA

By Tuomas Kuusela

Introduction

By generating a dataset with a function, which simulates data points distributed along two overlapping ellipses, we can pose a challenge for linear classification methods. Due to the non-linearities and the overlap in the data distribution, linear classifiers struggle to separate and classify the data points accurately. Initially it was demonstrated that it is possible to achieve a degree of separation using a non-linear transformation.

The primary objective shifted to the use of Kernel Principal Component Analysis (Kernel PCA) as an alternative method. Using the polynomial kernel function, we can project the dataset into a higher dimensional space, where the data relationships can be linearly separated more effectively. This approach showcases the adaptability and strength of Kernel PCA revealing the most informative principal component that encapsulates the dataset's variance.

Results

By applying a polynomial kernel to perform Kernel PCA on the dataset, we could uncover the linear separability that was not apparent in the original space. After choosing the proper kernel function, the steps taken are as follows:

1. **Computation of the Gram Matrix:** Calculating the mutual kernel values between observations using a polynomial kernel, denoted by $\kappa_\phi(x_i, x_j)$, and arranged these into a $n \times n$ matrix, K. This matrix known as the Gram matrix represents the inner products in the feature space induced by the polynomial kernel, encapsulating the enhanced relationships between datapoints in the transformed space.
2. **Centralization of the Gram Matrix:** The centralization was performed according to the formula $\mathrm{K}^* = \mathrm{K} - \mathrm{CK} - \mathrm{KC} + \mathrm{CKC}$, where C is a $n \times n$ matrix with each entry equal to $1/n$ and n is the sample size.
3. **Eigen Decomposition:** This process decomposes K^* into its eigenvalues and eigenvectors, revealing the principal components of the data in the transformed feature space. The focus lied on the third eigenvalue λ_3 and its corresponding eigenvector e_3, which were identified as being particularly informative for our analysis.
4. **Calculation of the Principal Component:** The most informative principal component was calculated by multiplying the third eigenvector e_3 by the square root of its corresponding eigenvalue $\sqrt{\lambda_3}$.

In Figure 1 below, we can see the progression from the original elliptical clusters through non-linear transformation to Kernel PCA separation. This proves the effectiveness of the Kernel Trick in uncovering hidden patterns within the data.

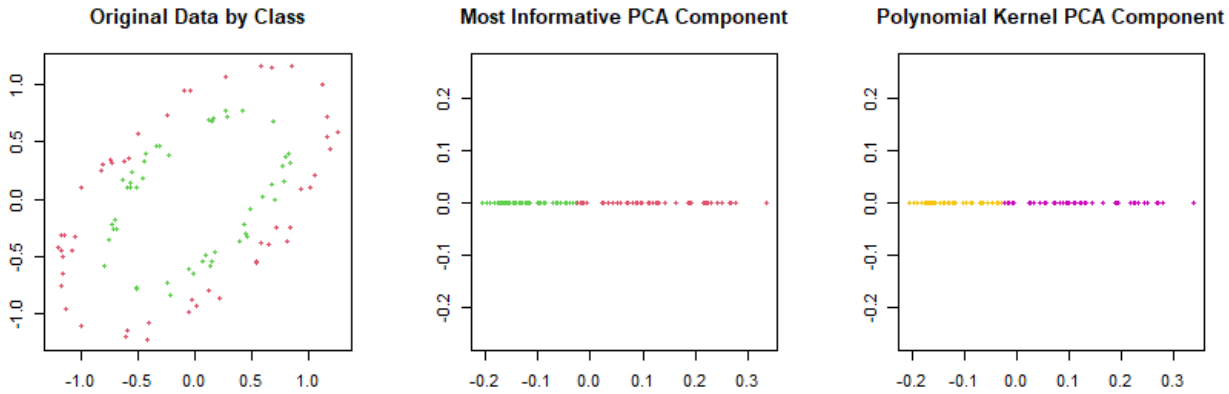

Figure 1. *Left: Original data display the elliptical clusters. Center: Non-linear transformation showing the initial separation. Right: Kernel PCA highlights its separation like the non-linear transformation.*

Image Classification

By Tuomas Kuusela

Introduction

This problem revolves around image classification using the MNIST dataset, a collection of Zalando's article images. The dataset contains 60000 training examples and 10000 test examples, each being a 28×28 pixel grayscale image labelled across 10 different classes. The objective is to train three different models, Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel, Artificial Neural Network (ANN), and Convolutional Neural Network (CNN), to classify the images and asses each model's performance.

For the SVM model, Principal Component Analysis (PCA) was used to reduce the dimensionality before training, enabling a more efficient use of the RBF kernel. The number of principal components (PCs) retained for the model was determined based on the cumulative variance explained, aiming for an optimal balance between information retention and computational efficiency. The model was finetuned with different gamma values to maximize its accuracy.

The ANN model was designed with three hidden layers, containing 128, 24, and 24 neurons while using dropout layers to mitigate overfitting. Meanwhile the CNN model incorporated convolution layer with 32 filters, followed by a pooling layer and a hidden layer with 100 neurons.

Results

SVM with RBF kernel

The initial steps involved in SVM modelling was PCA on the dataset provided to reduce the dimensionality of the feature space. In Figure 2 below, we can observe how the cumulative variance is explained as a function of the number of PCs. This step is critical in identifying the most informative features and thus enhancing the model's efficiency and performance.

The PCA results revealed that a significant portion of the variance could be explained by a subset of the PCs. To capture 90% of the variance, a substantial number of PCs was required, while around 70% and 80% of the variance could be explained with notably fewer components.

Based on the insight from PCA, 49 PCs were selected for the SVM model, as these explained 80% of the variance, establishing an optimal balance between information retention and computational efficiency.

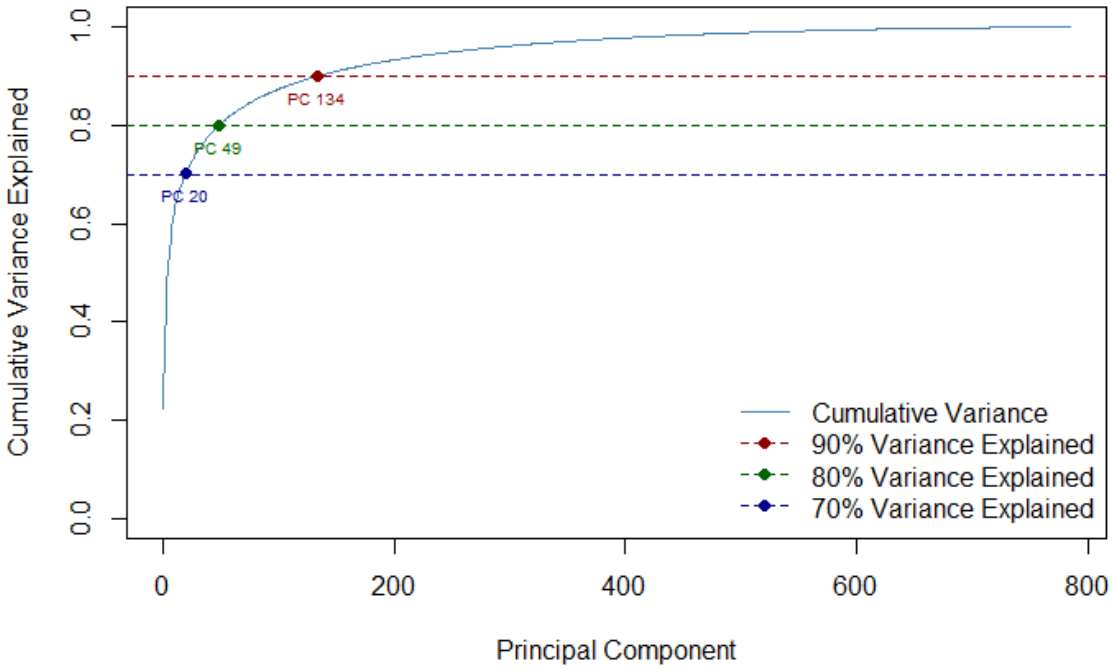

Figure 2. *Cumulative variance explained for specific PCs.*

The results of the SVM model achieved a total accuracy of 86.5% with a gamma value of 0.01. This performance is acceptable, especially considering the reduced dimensionality of the input data. The accuracy level for different numbers of PCs were also recorded, showing that the selection of 49 PCs was justified as it provided a good balance between dimensionality reduction and model accuracy.

In addition to selecting the optimal number of PCs, the gamma parameter was tuned to optimize the model's performance. Through systematic testing of various values, it was determined that a gamma value of 0.01 resulted in the highest overall accuracy.

Artificial neural network

The ANN model was designed with three hidden layers, following the suggested configuration of 128, 24, and 24 neurons. Dropout layers were incorporated with rates of 0.4 for the first layer and 0.3 for the latter two layers. This also involved reshaping the data input images from 28×28 pixels into a flat vector of 784 pixels and normalizing the pixel values to the range of 0-1.

Training the ANN model was initially planned for 50 epochs with a batch size of 128, using a validation split of 20% to monitor the model's performance on unseen data. Validation loss, indicating

the model's prediction error on the validation set, served as an indicator of the model's ability to generalize beyond the training data.

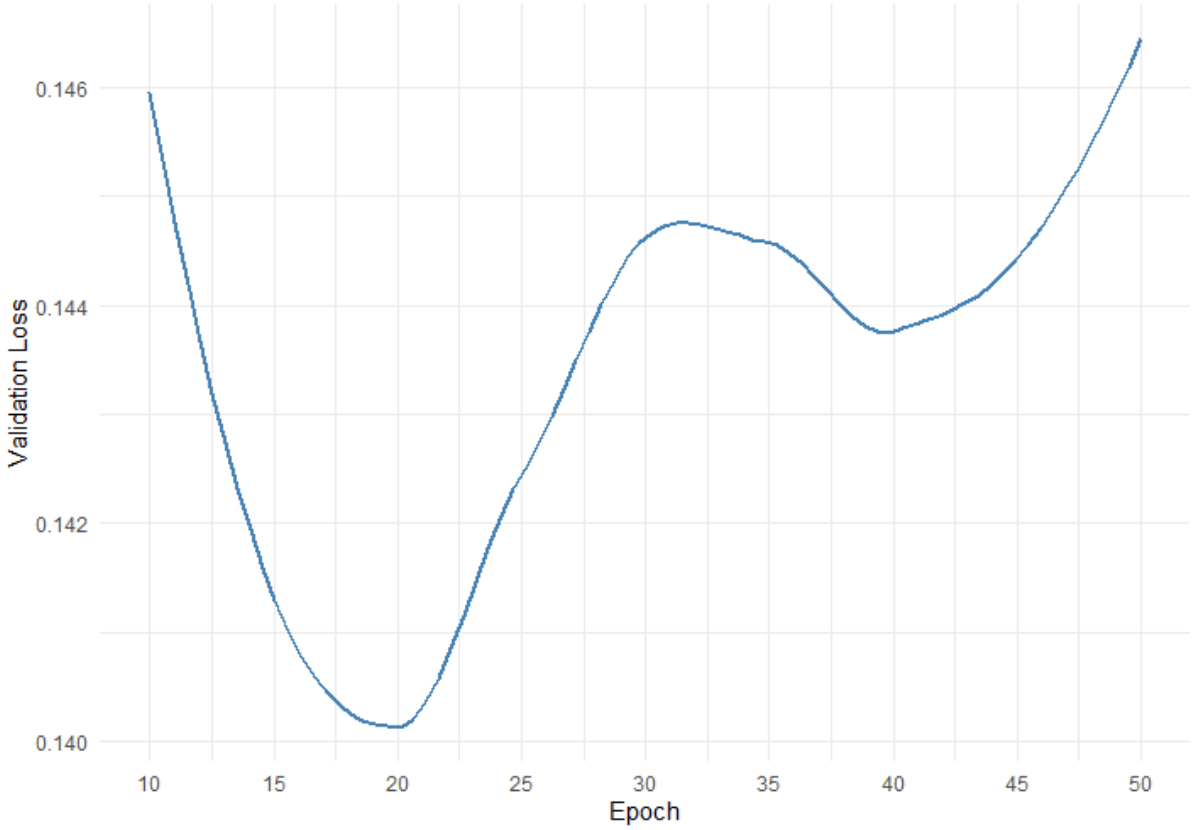

Figure 3. *ANN validation loss based on the number of epochs.*

Figure 3 presents the validation loss over epochs, which visualize the role in determining the most effective number of epochs for training. It became evident that beyond 20 epochs, the reduction in validation loss began to plateau and began to slightly increase. This suggests that the model started to overfit, learning the noise in the training data as if they were significant patterns, which detracts from its ability to generalize to new data. Therefore, the decision was made to limit the training to 20 epochs, aiming to balance between the learning process and preventing overfitting.

The final ANN model achieved an accuracy of 97%, showcasing robust capability to classify images effectively. It's worth noting that the model's performance could potentially be further optimized by experimenting with other hyperparameters such as batch size and validation split.

Convolutional Neural Network

The CNN model was designed with a convolutional layer equipped with 32 filters and 3 × 3 kernel size, followed by a 2 × 2 pooling layer. After the pooling layer, the data was flattened to transform the 2D feature maps into a 1D feature vector. A hidden layer with 100 neurons was included into the model.

Adopting a consistent approach with the ANN model, the CNN model utilized a batch size of 128 and a validation split of 20%. The decision to train the model for 10 epochs was determined by observations like those made for the ANN model, with emphasis placed on monitoring the validation loss to identify the optimal training duration, this is visualized in Figure 4 below.

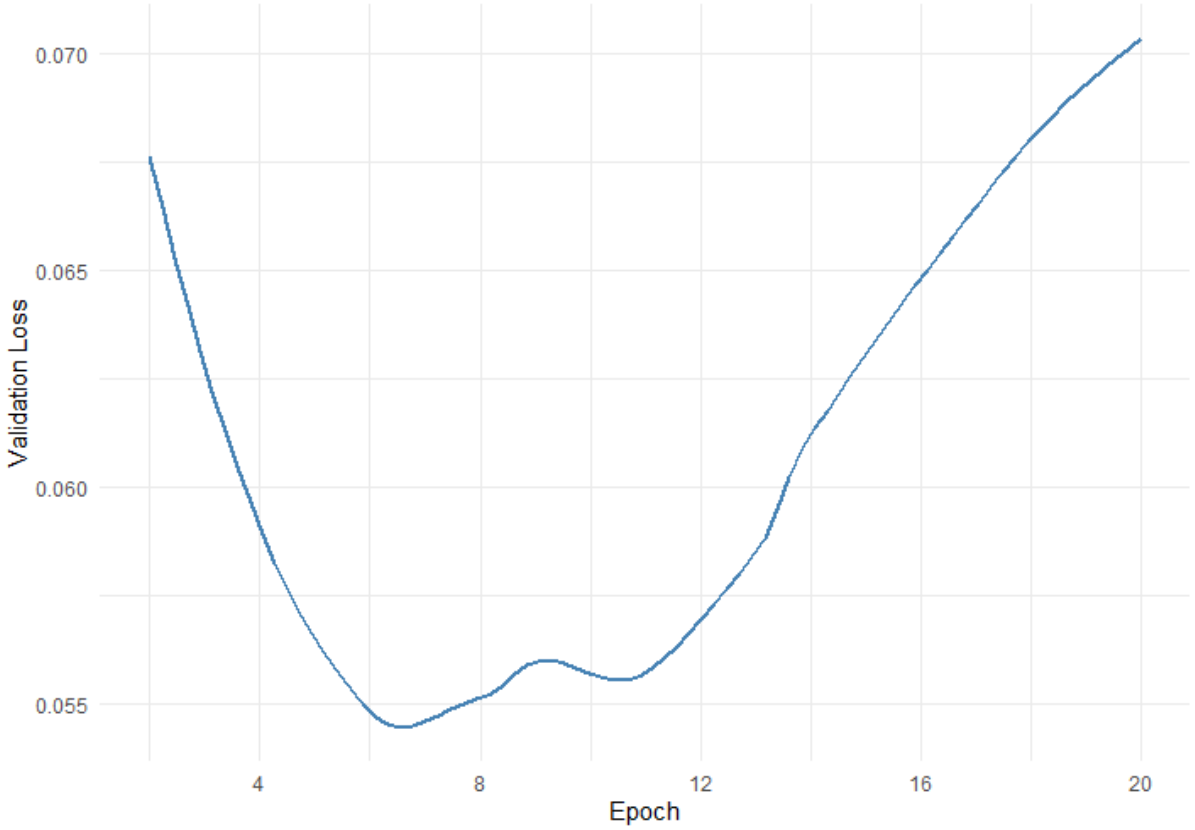

Figure 4. *CNN validation loss based on the number of epochs.*

The final CNN model achieved an accuracy of 98.5%, proving its capability in classifying images like the ANN model. With a slightly higher accuracy compared to the ANN model, it proves its strengths in image recognition tasks, which they are renowned for. Like the ANN model, there is room for finetuning with other hyperparameters such as batch size and validation split.

Discussion

The analysis provided insight into respective capabilities in image classification tasks for all three models. However, a crucial aspect to consider in this comparison is the nature of the datasets used for training and testing each model. The SVM model was trained and tested on a provided dataset, while the ANN and CNN models utilized the original dataset from the Keras R-package. This distinction in data sources introduces a notable variable into the evaluation of model performance.

The accuracy achieved by the SVM model, while lower compared to the ANN and CNN models, necessitates consideration of the dataset differences. It's important to acknowledge that these datasets, though similar in nature and content, are not totally identical. Such disparities can significantly impact model performance, as variations in data distributions, sample size, and even preprocessing steps can alter the difficulty level of the classification task. Therefore, direct comparison of accuracy figures between the models must be approached with caution. However, the ANN and CNN models can be compared with each other since the same dataset was used.

Additionally, it's worth mentioning that ANN and CNN are often considered better suited for image recognition tasks than SVM.