**GE-461 Introduction To Data Science**
**Project-2**

Mertbora Erbay 21703550

**Setup:**

In order to be able to take advantage of the massive catalogue of libraries, the project utilizes python. In order to be able to read the dataset given in ".mat" format, mat4py library is used and some data preparation is conducted. After that, "test_train_split" method of sklearn is used to create a training set by a random sample of 2500 digits and labels. In order to be able to ensure consistency, a constant seed of 45 was selected.

**Question 1:**

To apply PCA on the 400 dimension dataset, the plot of eigenvalues in decreasing order can be seen below. Observing this plot we can see that the optimal number of principle components would be between 50 and 100. I would choose 70 components looking at the plot.

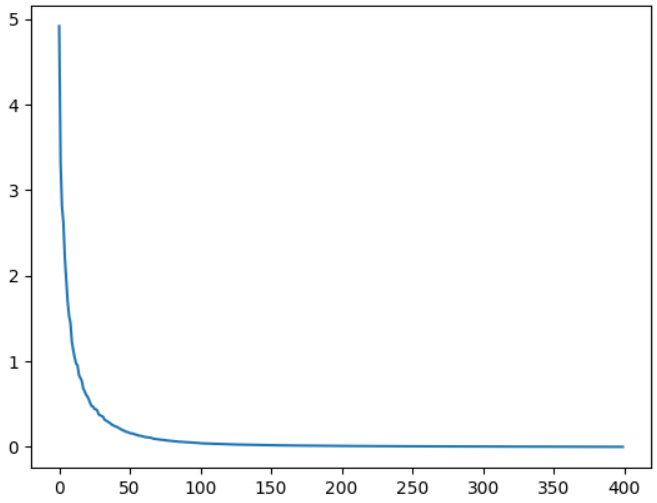


*Figure 1: Plot of Eigenvalues*

Using mean method of sklearn, the 20x20 element wise mean of all arrays in the feature set was calculated and this was converted to an image representing the average of all digits.

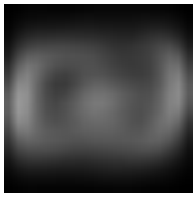

*Figure 2: 20x20 image of average of all digits*

*(the image was enhanced by MS Word automatically)*

After that, getting the first 10 eigenvectors calculated using PCA, they are converted to images. The resulting images can be seen in Figure 3.

*Figure 3: 20x20 images of average of each digit*

*(the images were enhanced by MS Word automatically)*

Observing the results, it was interesting to find out that some of the digits did represent the classes they were supposed to, they were rotated by 90 degrees. They should not represent the classes exactly since these are the highlights of the largest variances in the images. However, the eigenvectors does seem to be different which makes sense in order to be able to classify the classes correctly which suggests that this classification method is applicable for this case.

Remembering the eigenvalue compared to component count plot in the beginning, in order to be able to further observe the effect of number of components on the accuracy of the model, the model was trained using data resulting from PCA of 22 different component counts. In Figure 4, component count compared to percent ratio of correct classifications out of 2500 labels in the test sample using the test digits can be seen.

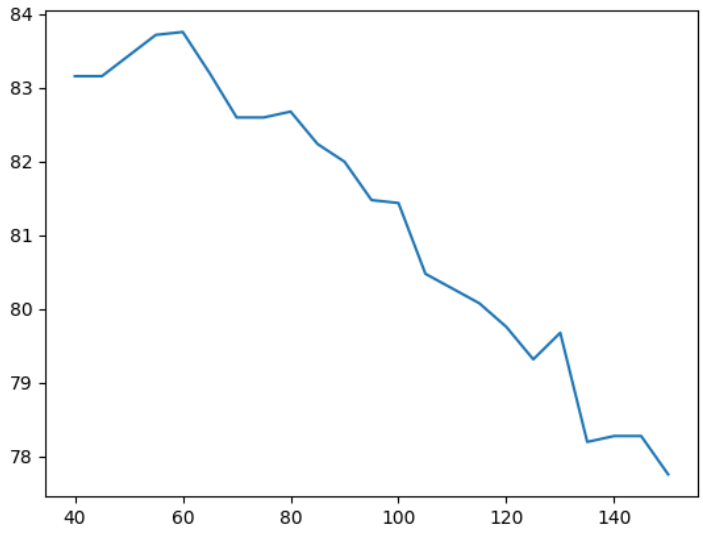


*Figure 4: component count on the x-axis compared to accuracy percentage on the y-axis*

Observing the plot we can see that the maximum accuracy is achieved at 60 principle components with a accuracy of 83.7%. This result falls in line with the observations from Figure 1.

**Question 2:**

Using “means_” attribute of “LinearDiscriminantAnalysis” function of sklearn, we obtain a new set of bases for the training dataset. The image reproduction of the bases obtained via LDA can be seen in Figure 5.

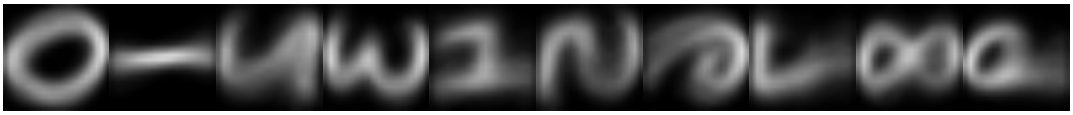

*Figure 5: New set of bases obtained via LDA in image form*

Although a different method was used to obtain the bases, we can see that these resemble the classes they are supposed to resemble. This makes sense since these are approximately the means of all digits for a label. My expectations were that these would be negative images of the ones from Figure 3 though it seems that these are more representative of the classes.

Using subspaces of dimensions ranging from 1 to 9 and training a Gaussian classifier using data for each dimension of subspaces, the following plot of success-rate (percentage of successful classifications as a percentage of 2500 test samples) compared to dimensions is obtained.

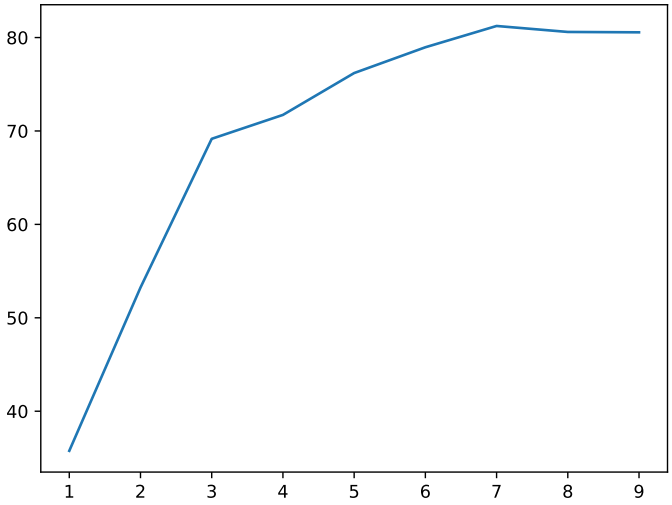


*Figure 6: LDA dimension in x-axis, success rate in y-axis*

We observe the highest success rate of 81.1% with a dimension of 7 and we can observe that generally speaking, the success-rate increases as the dimension increases. Considering LDA is a way of discriminating components while PCA is way of observing which components are more effective. Therefore, the results make sense since dimension of subspaces in LDA and number of components in PCA should be inversely proportional at similar success rates.

Comparing the maximum success-rate of the Gaussian classifiers using PCA and LDA, we can see that PCA is more successful by a small margin. This margin is very small and may be attributed to the randomly selected training sample favoring PCA or the fact that, due to the high dimensionality of the problem, more fine-tuning of the component count is possible when using PCA and hence, in LDA, the highest possible success rate may fall between 6 and 7 or 7 and 8 dimensions of subspaces. Also, considering this dataset is of handwritten digits, there may be differences between the way people write the same digits, since PCA is a linear transformation technique that tries to find the components with the highest variance, it is great tool that can differentiate writing errors and principle components that affect the results because it does not take the labels into account. However, LDA takes labels into account while trying to discriminate the worse components. This means writing errors play a larger role in LDA. One example of this may be the different ways the digit 4 can be written. If the problem was to classify more universally accepted labels, LDA may have been more successful.

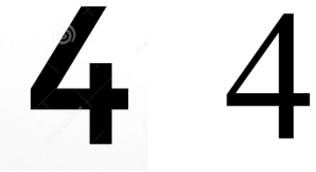

*Figure 7: Two different forms of the digit 4*

**Question 3:**

Using sammon repository found online to use for Sammon's Mapping, we can observe that after 107 iterations, the error converges at almost 0.067. Using Sammon's Mapping, the following scatter plot is obtained.

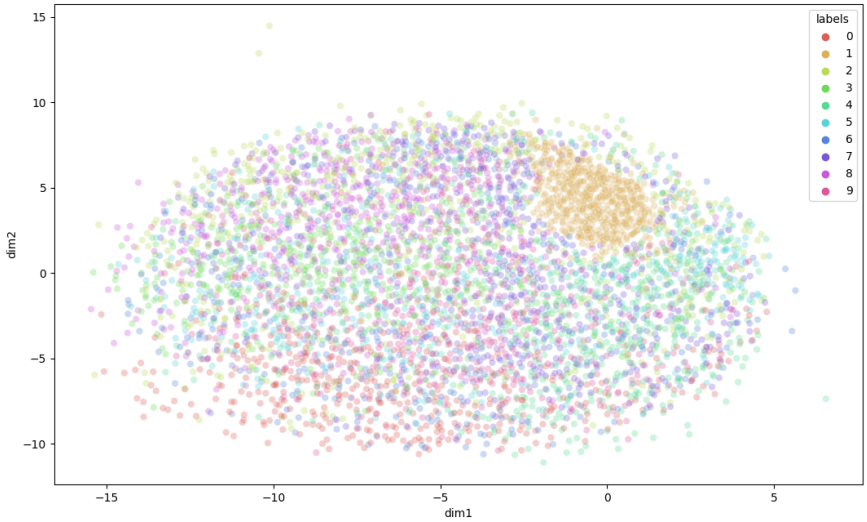


*Figure 8: Sammon's Mapping*

Looking at the scatter plot, other than the digit 1, we can see that the plot does not seem to reflect the classes in a discernable way.

Using sklearn's t-sme method, t-sme was applied to the training dataset. To be able to plot the scatter, t-sme with 2 dimensions was utilized. After experimenting with different perplexity values, the maximum perplexity value of 50 seemed to yield the best results in terms of separating the classes as visibly as possible. Trying different number of iterations, namely, 250,400,500,750 and 1000. The best plot was achieved using 50 perplexity with 500 iterations. The plots for each can be seen in the Figures below.

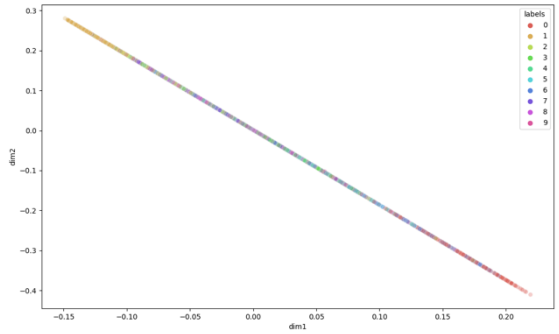


*Figure 9: t-SNE with 50 perplexity 250 iterations*

250 iterations seem to result in a linear scatter, which is not ideal since it is difficult to differentiate the labels using this graph seen in Figure 7.

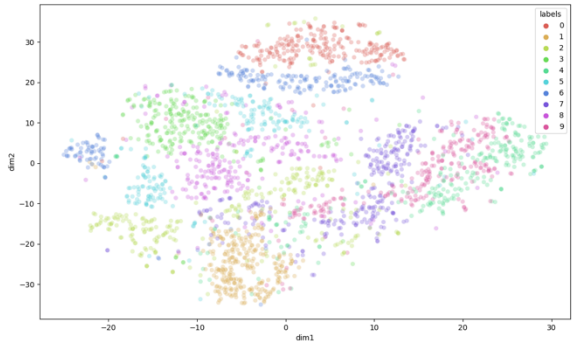


*Figure 10: t-SNE with 50 perplexity 400 iterations*

400 iterations seem to be able to distinguish between the labels better than 250 iterations, with 50 perplexity.

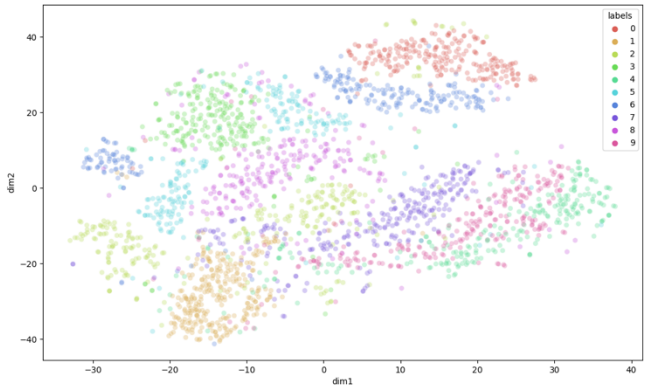


*Figure 11: t-SNE with 50 perplexity 500 iterations*

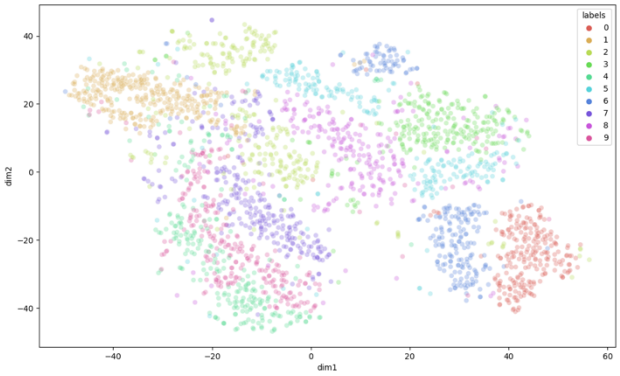


*Figure 12: t-SNE with 50 perplexity 1000 iterations*

There does not seem to be major differences in terms of differentiating between 1000 and 500 iterations with 50 perplexity, however 500 iterations seem to result in a more spread out scatters between labels.

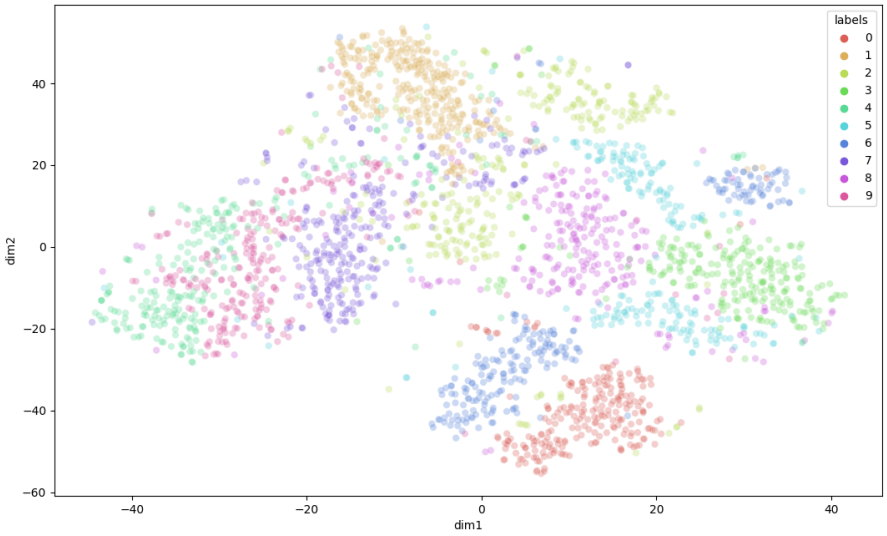


*Figure 13: t-SNE with 50 perplexity 500 iterations*

Finally, we can see that 50 perplexity and 500 iterations yields the best results. We can also see how 3 and 8 scatters are close by while 1 and 0 are far apart. This makes sense since 3 and 8 is similar while 1 and 0 are not.

**References:**

Mat4py library: https://pypi.org/project/mat4py/
Sklearn library: https://scikit-learn.org/stable/
Sammon: https://github.com/tompollard/sammon
Seaborn library: https://seaborn.pydata.org/