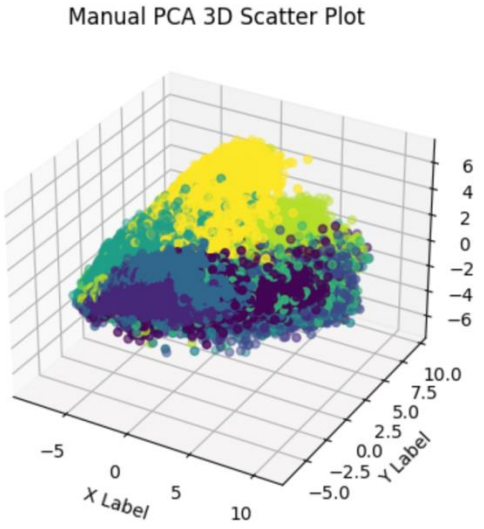
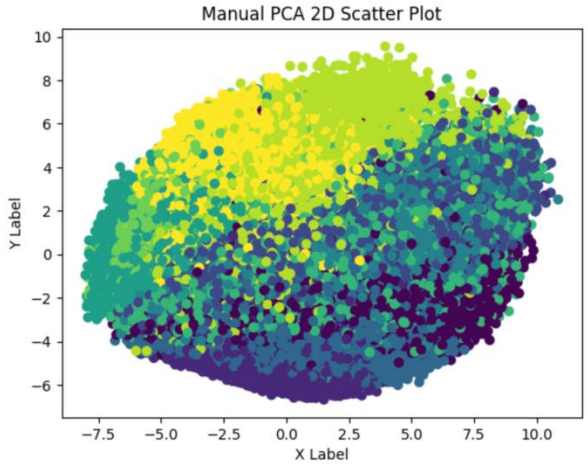
First we start by loading our train data in X_train and y_train, and then we load our test data in X_test and y_test. By watching our data we find that our train data has 60000 samples and 784 features (which are pixels of a 28*28 picture) and a label from 0 to 9.

After that we start some checking to see if our data needs any preprocessing. First we check if we have any missing values and thankfully there isn't any. After that we check pixels interval and we see that it's from 0 to 255 which is very good but for caution we divide it by 255 to make it in [0, 1]. After that we check for imbalancement and again thankfully we have 6000 samples from each label.

After that it's time for implementing PCA, and we do as said in PowerPoint of chap 4. After implementing PCA we want to display after transformation, but to display data we should have at most 3 dimensions, so we fit our PCA once with 2 eigenvectors selected and once with 3 eigenvectors selected and transform our data based on both of them and plot them.





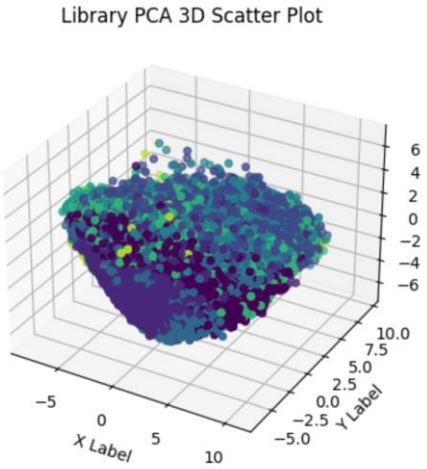
As we can see both 2D and 3D PCAs has separated our data a little, but it's natural that with loosing 782 features we won't have a good separation.

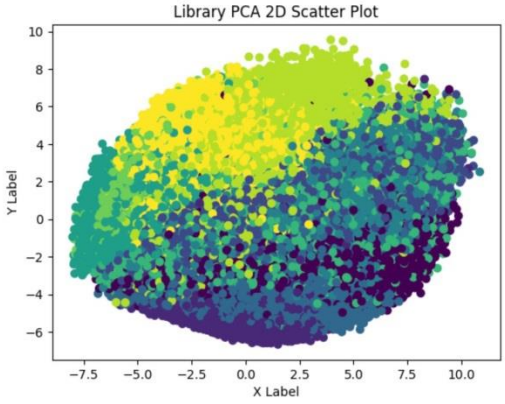
In generalization, principal components are directions which maximize variance among all directions and each one is orthogonal to other ones and when we choose some of them, we choose the ones with maximum eigenvalue. So if I want to say what does this principle components show, I would say that they show top 3 or 2 orthogonal directions which maximize variance and saves us the most data from other directions. And for contribution of each feature and each principle component we must find each features value on each eigenvector and the feature with most value has the most contribution so for our 3 principle components we have that:

1. maximum contribution with first principle component is with feature pixel151
2. maximum contribution with second principle component is with feature pixel443
3. maximum contribution with third principle component is with feature pixel399

(for watching all contribution ratios, you can go to main.ipynb)

Now we use sklearn.decomposition.PCA library to calculate 2D and 3D PCAs again and compare them with our PCAs. First let's see their plots:





In first sight we can see that our 2D PCA and scikit's 2D PCA are really close but there is a lot of difference in 3D PCAs. The fact is they both are correct and very similar but difference comes from that some of principle components are negative of the ones in other PCA and that's why plots are very different. Approximately you can say that second plot is the first one rotated, but it's not completely correct because all of the principle components are not negative of the one in other PCA.

Know we start to learn best parameters for SVM with 5-fold classification and our learning parameters are:

1. Number of PCA components: with increasing number of components we have more data but it takes much more time to learn the model and we have to limit it because of our computation limits.
2. Sample size: with increasing number of components we have more data but it takes much more time to learn the model and we have to limit it because of our computation limits.
3. Kernel type:
    a. Linear: Forms linear decision boundaries in the input space.
    b. Poly: Allows for curved decision boundaries using polynomial functions and we choose it's degree for more specification.
    c. Rbf: Creates non-linear decision boundaries using Gaussian-like functions.
    d. Sigmoid: Maps the input space into a hyperbolic tangent function for non-linear classification.

After learning we find that best parameters are:

```
best_score 0.8618500000000001
best_Sample_Size 20000
best_Kernel rbf
best_PCA_component 40
best_Degree 0
```

(and you can see accuracy, precision, recall and f1_score for each model in main.ipynb)

It shows that more data will give us more accuracy in this model and also because the best model is not polynomial our degree is zero.

Know we start to learn best parameters for Naïve Bayes with 5-fold classification and our learning parameters are:

1. Number of PCA components: with increasing number of components we have more data but it takes much more time to learn the model and we have to limit it because of our computation limits.

After learning we find that best parameters are:

```
best #pca component :  40
best score:  0.773
```

(and you can see accuracy, precision, recall and f1_score for each model in main.ipynb)

And we can see that the most PCA component doesn't give us the best accuracy because:

1. PCA reduces the dimensionality of our train data while preserving important information. This reduction helps us to scape curse of dimensionality and overfitting
2. Our noise would be reduced and our model will work much better
3. Classes will get closer to each other and probabilities gets better

But also very small number of components is not good because we will lose a lot of information.

Know we start to learn best parameters for KNN with 5-fold classification and our learning parameters are:

1. Number of PCA components: with increasing number of components we have more data but it takes much more time to learn the model and we have to limit it because of our computation limits.
2. Number of neighbors: it shows that our model looks for how many neighbors to classify.
3. Weights:
    a. Uniform: All neighboring points have an equal contribution to the classification decision.
    b. Distance: Neighboring points have weights inversely proportional to their distance from the query point, giving closer points more influence.

After learning we find that best parameters are:

```
best #pca component :  160
best score:  0.8632666666666665

best N neighbor:  5
best Weight: distance
```

(and you can see accuracy, precision, recall and f1_score for each model in main.ipynb)

As you can see here again PCA helps us to avoid noises and overfitting and also number of neighbors also says that if we fix small amount of neighbors we will have overfitting and if we use large number of neighbors we won't be accurate.

Know we start to learn best parameters for MLP with 5-fold classification and our learning parameters are:

1. Number of PCA components: with increasing number of components we have more data but it takes much more time to learn the model and we had to limit it to 10 because of our computation limits.
2. Hidden layers: here we fix number of layers and size of them, but again because of computation limits we choose only one layer and two layer options.
3. Learning rates:
    a. Constant: Uses a constant learning rate throughout the training process, which may lead to slower convergence or overshooting in some cases.
    b. Adaptive: Adjusts the learning rate dynamically based on the progress of the training, such as reducing the learning rate when approaching a minimum or increasing it for faster initial learning
    c. Invscaling: Gradually decreases the learning rate over time based on the inverse of a scaling factor, aiding convergence and stability during training.
4. Activation functions:
    a. Logistic: Maps the input to a range between 0 and 1, resembling a logistic function, commonly used in binary classification problems.
    b. Identity: Provides a linear mapping where the output is equal to the input, often used in regression problems or as a pass-through activation function.
    c. Relu: Sets all negative inputs to zero and keeps positive inputs unchanged, commonly used in deep learning models to introduce non-linearity.
    d. Tanh: Rescales the input to a range between -1 and 1, similar to the logistic function but symmetric around zero.
5. Batch size: Refers to the number of training examples used in each forward and backward pass to update the model's parameters but because of computation limits we have avoided it.
6. Maximum number of iterations: Because of computation limits we have avoided it.

After learning we find that best parameters are:

```
best Hidden layer size:  (100, 25)
best Learning rate:  adaptive
best Activation:  relu
best score:  0.8282833333333333
```

(and you can see accuracy, precision, recall and f1_score for each model in main.ipynb)

It shows that here 2 layers work better than one layer and adjusting learning rate adaptive works and using relu activation function works better.

Now we learn our best models and calculate accuracy, precision, recall, f1_score and confusion matrixes for them.

SVM:

```
accuracy 0.8663
accuracy_train 0.88185
precision 0.8655349655632442
precision_train 0.8812240988809468
recall 0.8663000000000001
recall_train 0.88185
f1 0.865606765514738
f1_train 0.8811899794405045
confusion [[837   0  15  36   0   3  98   0  11   0]
 [  6 962   7  20   0   1   4   0   0   0]
 [ 11   1 788  12 103   0  78   0   7   0]
 [ 33   8   8 901  23   0  24   0   3   0]
 [  1   0  64  33 829   0  70   0   3   0]
 [  1   0   0   1   0 917   1  53   7  20]
 [165   1  97  35  64   0 628   0  10   0]
 [  0   0   0   0   0  31   0 899   0  70]
 [  5   1   8   3   1   5   8   3 964   2]
 [  0   0   0   0   0  18   0  44   0 938]]
confision_train [[1741    0   21   74    2    1  148    0   13    0]
 [  10 1920    5   52    5    0    7    0    1    0]
 [  12    0 1624   21  200    1  133    0    9    0]
 [  51    9   10 1821   54    0   49    0    6    0]
 [   2    1  147   88 1616    0  136    0   10    0]
 [   1    0    0    0    0 1880    0   77    6   36]
 [ 281    1  197   52  138    0 1312    0   19    0]
 [   0    0    0    0    0   46    0 1869    3   82]
 [   2    1    7    7    5    3   22    4 1946    3]
 [   0    0    0    0    0   27    0   65    0 1908]]
```

As we can see because of 5-fold cross validation rates are very close and we can say we have escaped from overfitting. Also by watching confusion matrixes we can say that labels 0, 2, 4 have the most mistakes in test and labels 2, 4, 6 have the most mistakes in train.

Naïve Bayes:

```
accuracy 0.7738
accuracy_train 0.7739833333333334
precision 0.774289079913175
precision_train 0.7746355572136405
recall 0.7738
recall_train 0.7739833333333335
f1 0.7734295173802775
f1_train 0.7736139761684246
confusion [[754   0  30  83   5  10  63   0  55   0]
 [  5 905  24  44   1   2  11   0   8   0]
 [  9   0 644   3 138  18 160   0  28   0]
 [ 60   4  12 834  30   7  47   0   6   0]
 [  2   0 106  44 695   3 143   0   7   0]
 [  0   0   0   0   0 797   7 127  32  37]
 [227   0 113  51  77   6 492   0  34   0]
 [  0   0   0   0   0  52   1 850   4  93]
 [  7   0  14   6   3  34  52  12 870   2]
 [  0   0   0   0   0  37   2  59   5 897]]
confision_train [[4685    1  148  466   33   37  410    0  220    0]
 [  53 5400  123  293   23    7   51    0   50    0]
 [  62    0 3827   31  831   66 1039    0  143    1]
 [ 407   19   62 4941  177   26  319    0   49    0]
 [  46    2  668  320 4009   10  871    0   74    0]
 [   7    0    1    3    0 4817   38  722  198  214]
 [1213    2  819  261  505   45 2896    0  258    1]
 [   0    0    0    0    0  337    6 5184   18  455]
 [  50    0   98   42   35  236  231   67 5230   11]
 [   2    0    2    1    0  167   10  327   41 5450]]
```

As we can see because of 5-fold cross validation rates are very close and we can say we have escaped from overfitting. Also by watching confusion matrixes we can say that labels 2, 5, 6 have the most mistakes in test and labels 5, 8, 9 have the most mistakes in train.

KNN:

```
accuracy 0.8704
accuracy_train 1.0
precision 0.8712163873783805
precision_train 1.0
recall 0.8703999999999998
recall_train 1.0
f1 0.8698504472140108
f1_train 1.0
confusion [[859   1  15  13   5   0  98   2   7   0]
 [  3 972   3  15   1   0   6   0   0   0]
 [ 18   0 783  12 105   0  78   0   4   0]
 [ 23   7  12 899  36   0  22   0   1   0]
 [  2   0  72  23 822   0  80   0   1   0]
 [  0   0   0   0   0 870   3  69   4  54]
 [192   1  88  17  76   0 616   0  10   0]
 [  0   0   0   0   0   4   0 946   0  50]
 [  2   0   8   1   5   1   5   3 973   2]
 [  0   0   0   0   0   4   0  32   0 964]]
confision_train [[6000    0    0    0    0    0    0    0    0    0]
 [   0 6000    0    0    0    0    0    0    0    0]
 [   0    0 6000    0    0    0    0    0    0    0]
 [   0    0    0 6000    0    0    0    0    0    0]
 [   0    0    0    0 6000    0    0    0    0    0]
 [   0    0    0    0    0 6000    0    0    0    0]
 [   0    0    0    0    0    0 6000    0    0    0]
 [   0    0    0    0    0    0    0 6000    0    0]
 [   0    0    0    0    0    0    0    0 6000    0]
 [   0    0    0    0    0    0    0    0    0 6000]]
```

As we can see because of nature of KNN, it's accuracy in train is 100%. Also by watching confusion matrixes we can say that labels 0, 3 have the most mistakes in test.

MLP:

```
accuracy 0.8745
accuracy_train 0.9978333333333333
precision 0.8751097508204515
precision_train 0.9978348317908463
recall 0.8744999999999999
recall_train 0.9978333333333333
f1 0.874674176261869
f1_train 0.9978330435451295
confusion [[813   1  19  21   6   3 130   0   6   1]
 [  5 985   1   6   1   0   1   0   1   0]
 [ 27   2 778   9  93   0  85   0   6   0]
 [ 30  13  13 883  23   1  33   0   4   0]
 [  3   3  77  34 811   0  70   0   2   0]
 [  4   1   0   1   0 921   0  41   9  23]
 [122   5  71  28  66   1 698   0   9   0]
 [  0   0   0   0   0  13   0 948   0  39]
 [ 13   0   6   5   2   7  10   2 954   1]
 [  0   0   0   2   0   6   0  36   2 954]]
confision_train [[5978    0    0    3    1    0   18    0    0    0]
 [   0 6000    0    0    0    0    0    0    0    0]
 [   4    0 5975    1   11    0    9    0    0    0]
 [   0    2    0 5998    0    0    0    0    0    0]
 [   1    0    7    5 5982    0    5    0    0    0]
 [   0    0    0    0    0 5975    0   20    0    5]
 [   9    0    5    3   16    0 5966    0    1    0]
 [   0    0    0    0    0    0    0 5998    0    2]
 [   0    0    0    1    0    0    0    1 5998    0]
 [   0    0    0    0    0    0    0    0    0 6000]]
```

As we can we have a overfitting in our model but it's still predicting well for our test data, so we can overlook it. Also by watching confusion matrixes we can say that labels 0, 6 have the most mistakes in test.

And by converting all of our prediction we can say that:

1. Labels 0 and 2 and 6 have the most mistakes and where the least recognizable ones.
2. KNN and MLP where the best models and if we had more resources MLP could have been much better. So we can say with less resource KNN is better for us and with better resources MLP will be better.

Ashkan Zarkhah

610399196