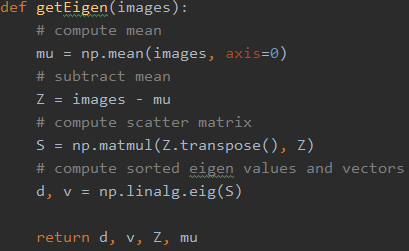
**Code explanation:**

**For PCA**, we created a function to get the eigen values and vectors of a given dataset, as explained in the lectures.

The eigen vectors will be sorted using the eigen values indices.



After, we create a new Matrix E, that corresponds to the best K vectors, with given number of samples, and multiply it with Z (which is the original images less the mean) to get the new feature vectors.

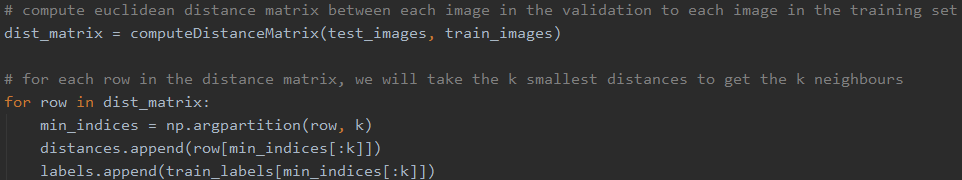


**For KNN,** we created **a multi-threaded version**, to increase performance.

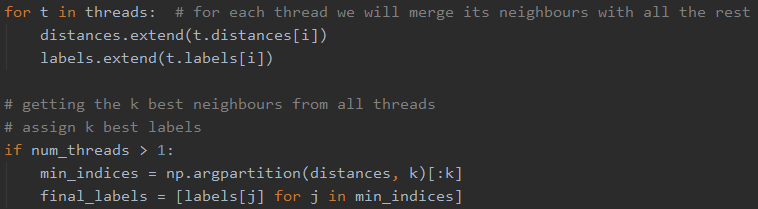
We first split the training set to equal ranges, as number of threads.



After, for each thread, we compute its K nearest neighbors with given range of indices.



In the end, we combine all the neighbors from all the threads and find the global K nearest for a given sample.



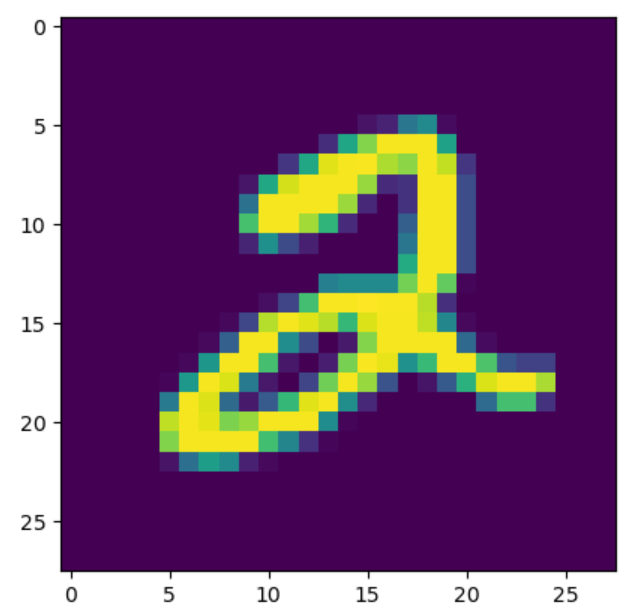
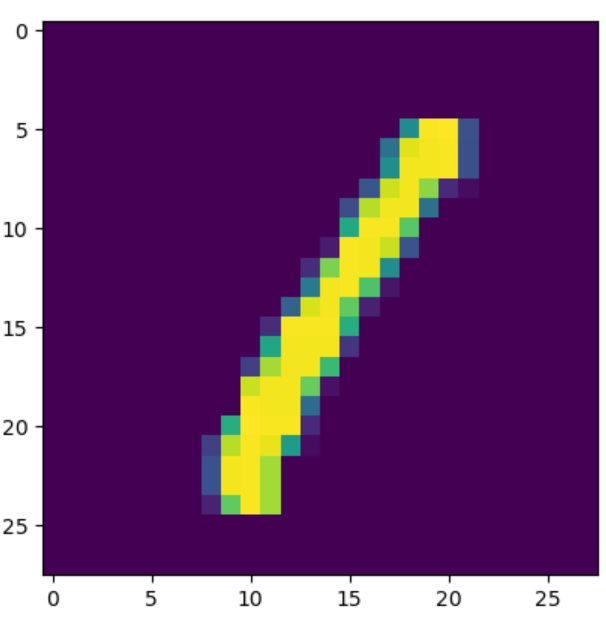
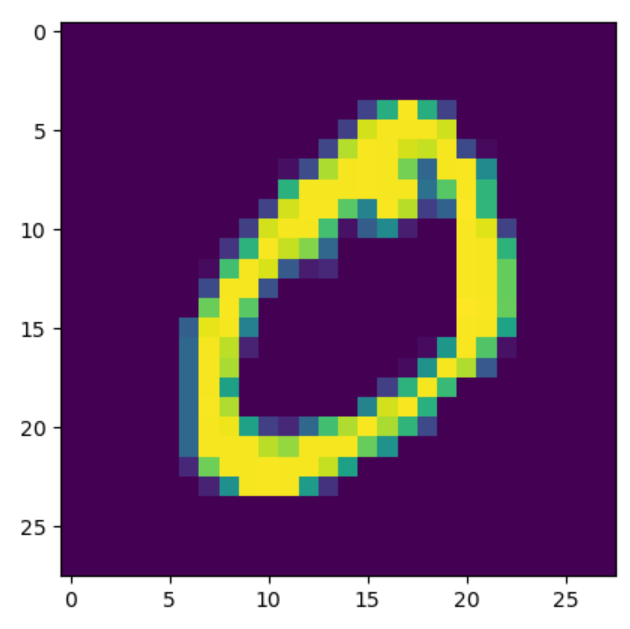
To compute the K-nearest, we create distance matrix where each row in the matrix represents validation\test sample, and each column represent its Euclidean distance from training set in the current index.

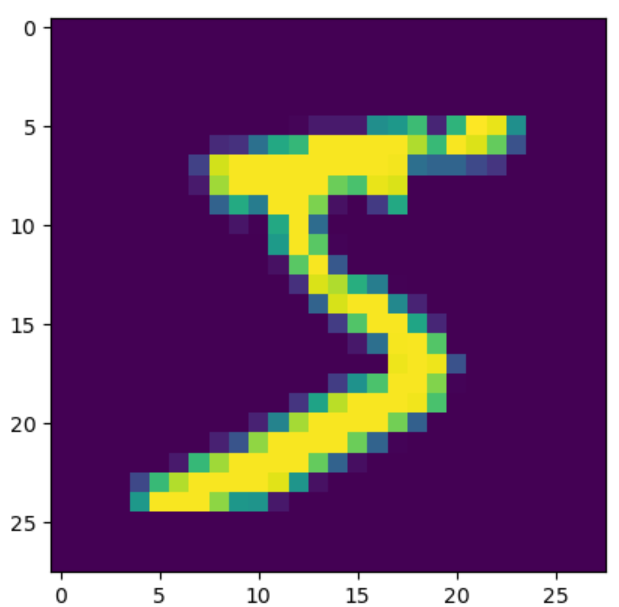
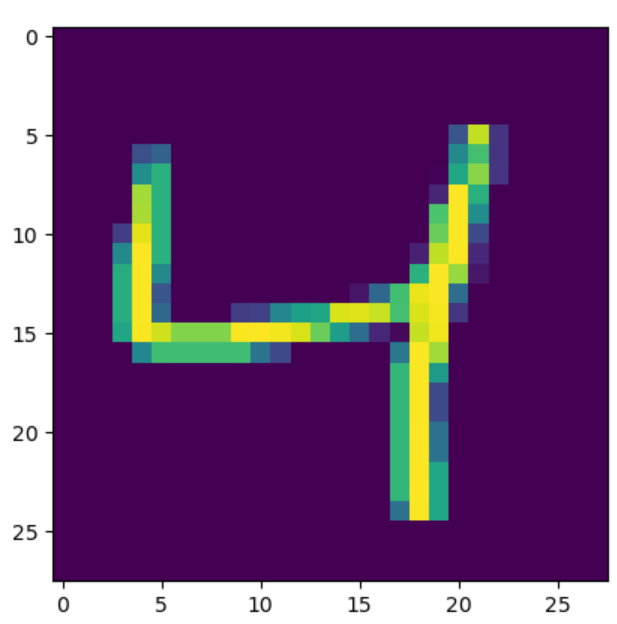
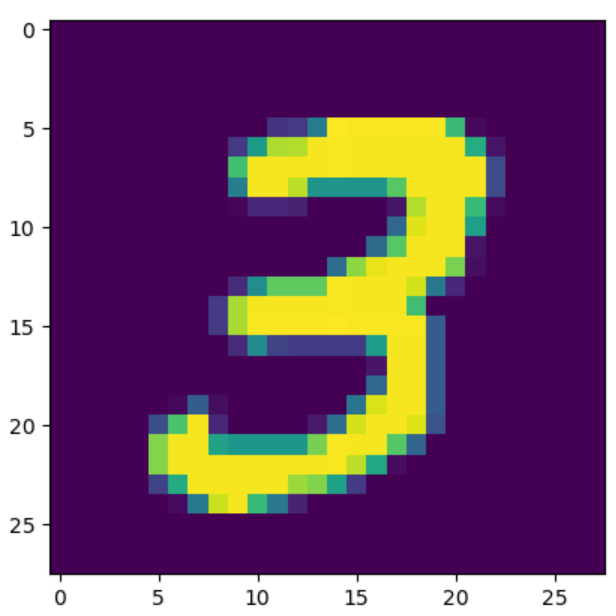
We created this matrix to avoid loops in computations, and **by using vectorized computation**, the running time decreased dramatically.

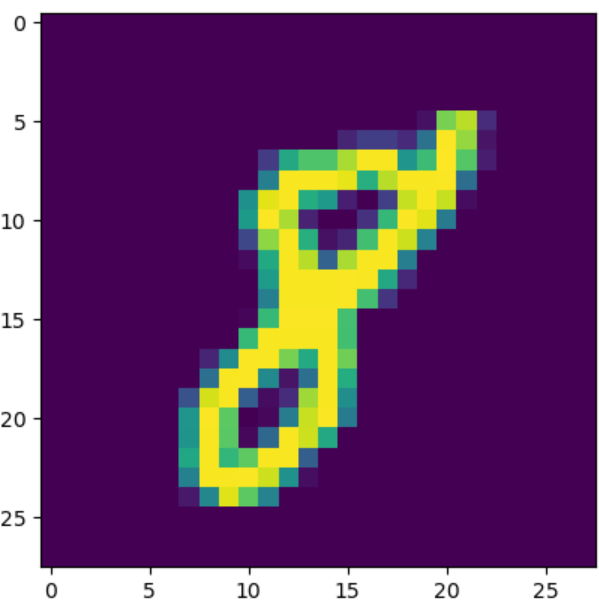
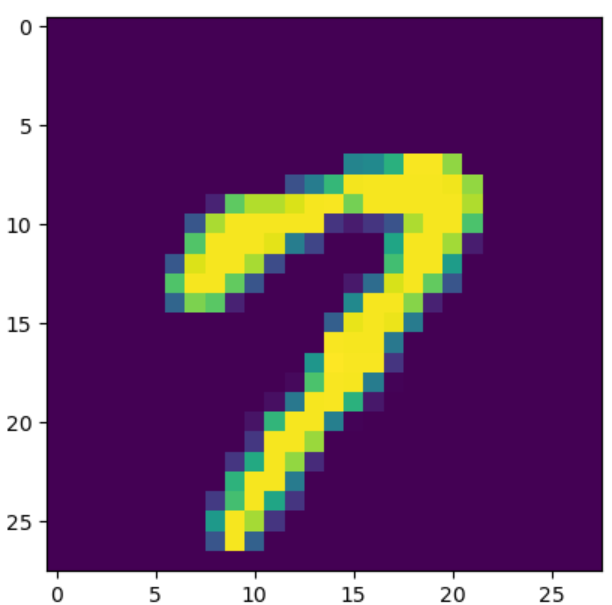
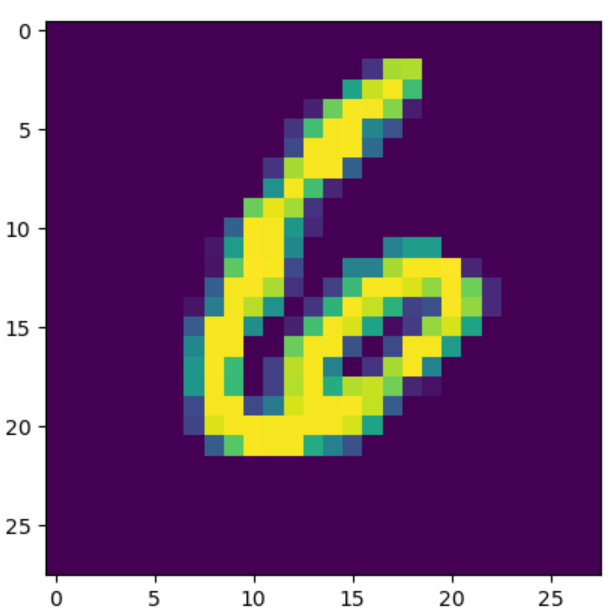
**לעבור על דרישות התרגיל שוב:**

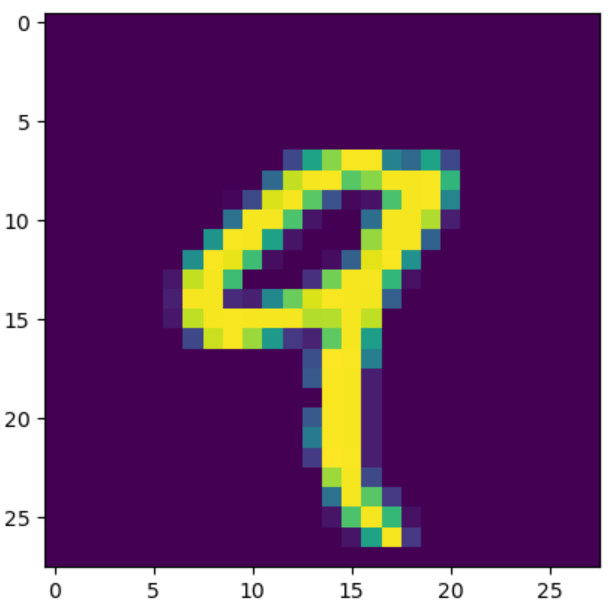
**Question 1:**

Step 1: Copy plots of each class (0 to 9)

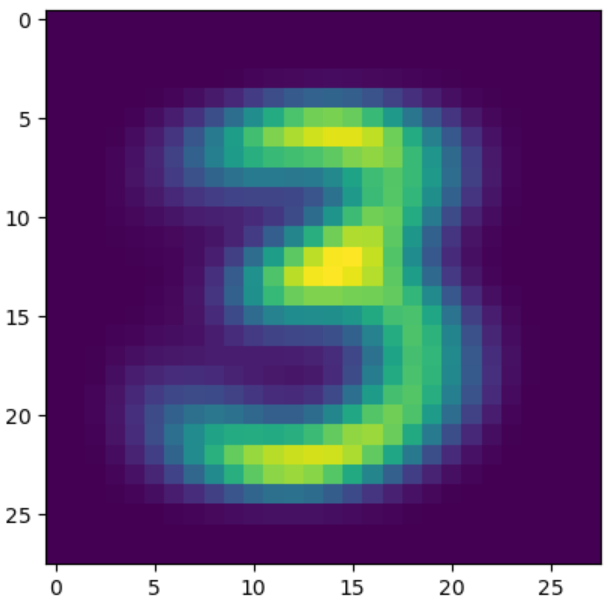
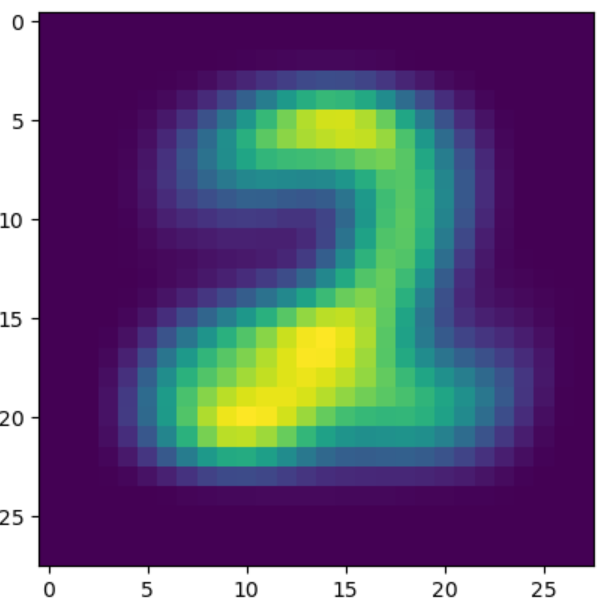
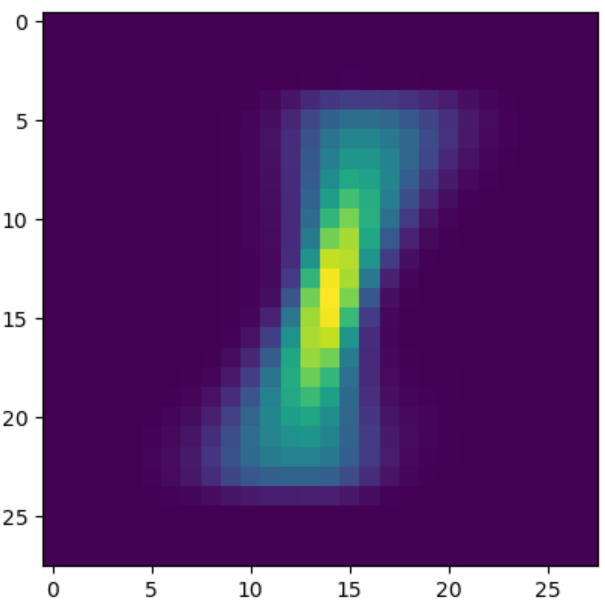
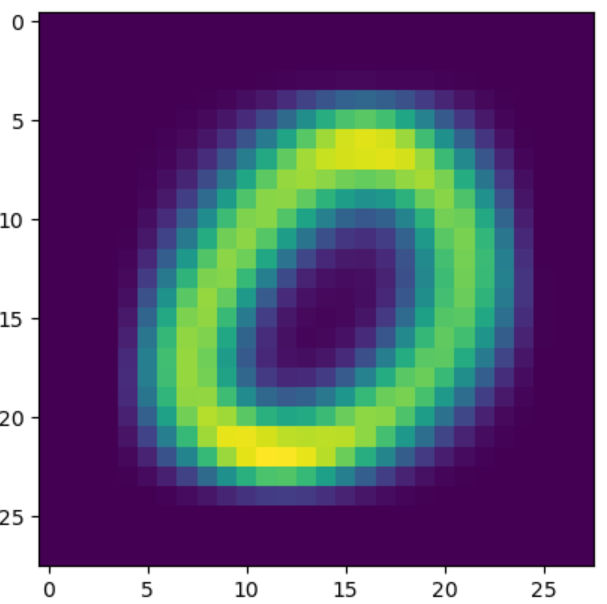


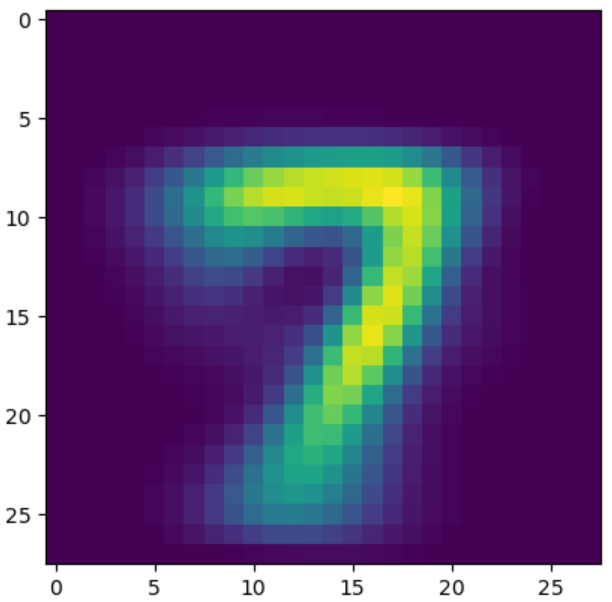
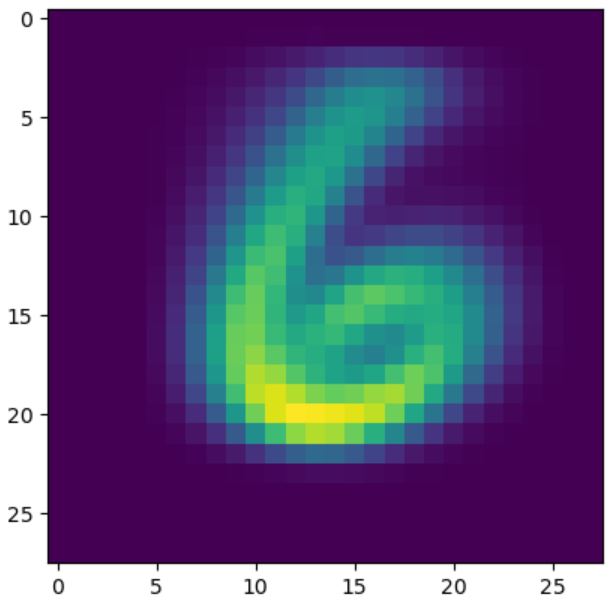
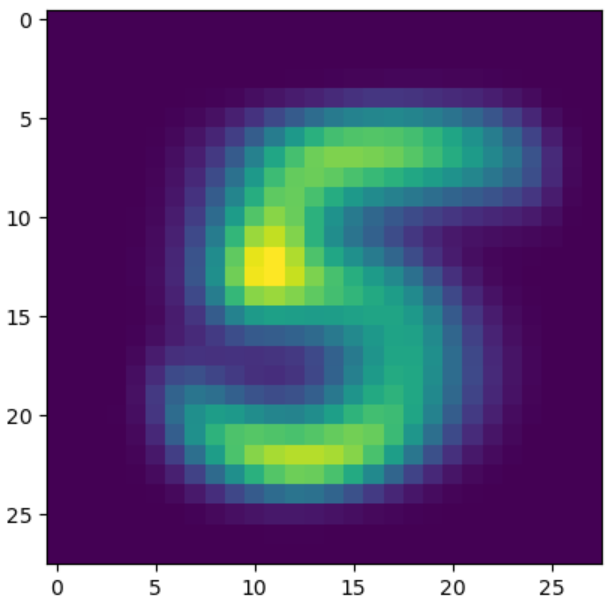
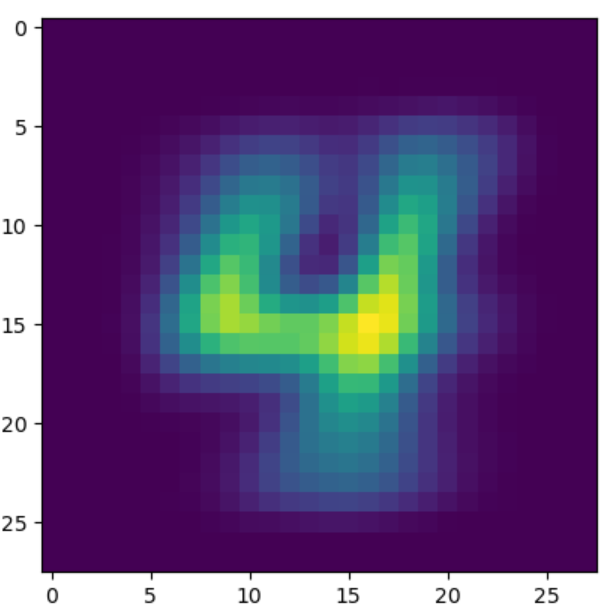


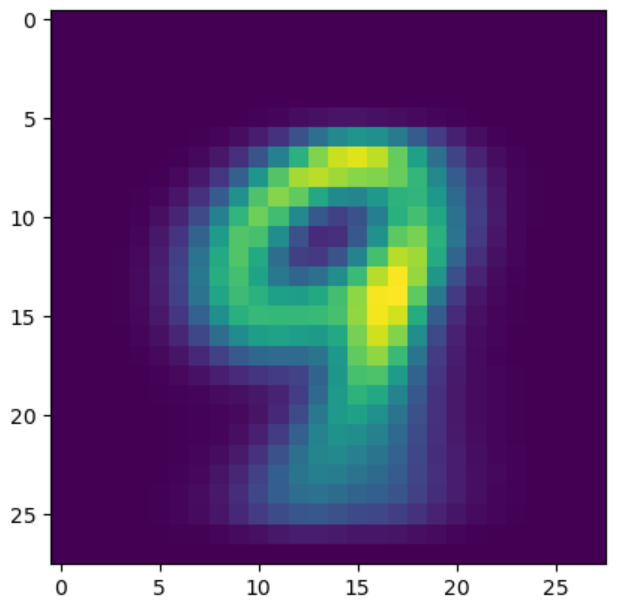
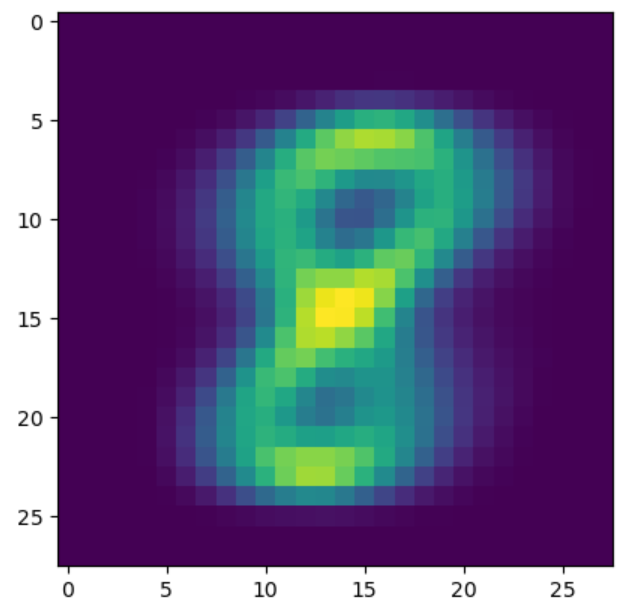




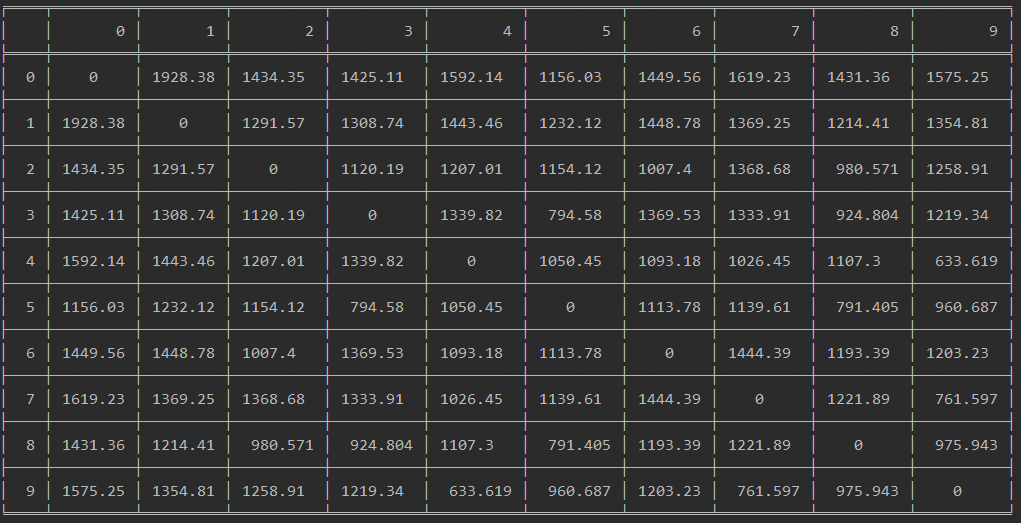
**Step 2**: Copy plots of each centroid (0 to 9)





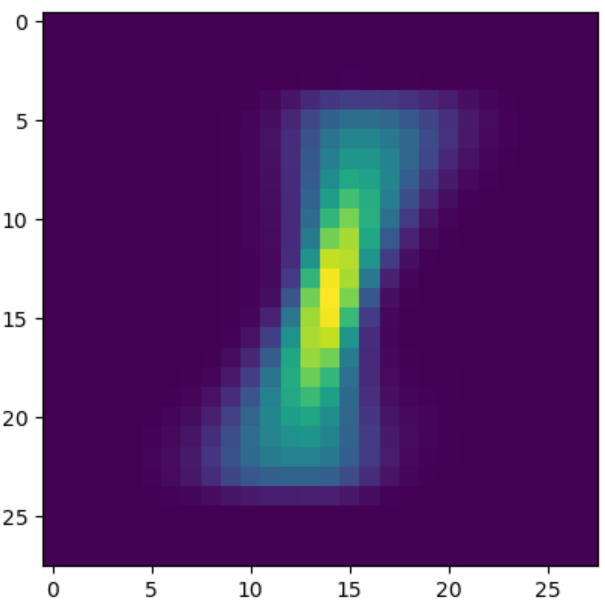
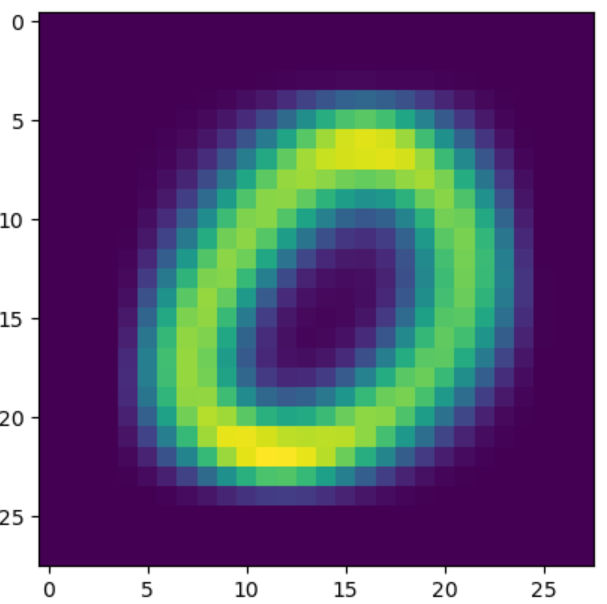


**step 3:**

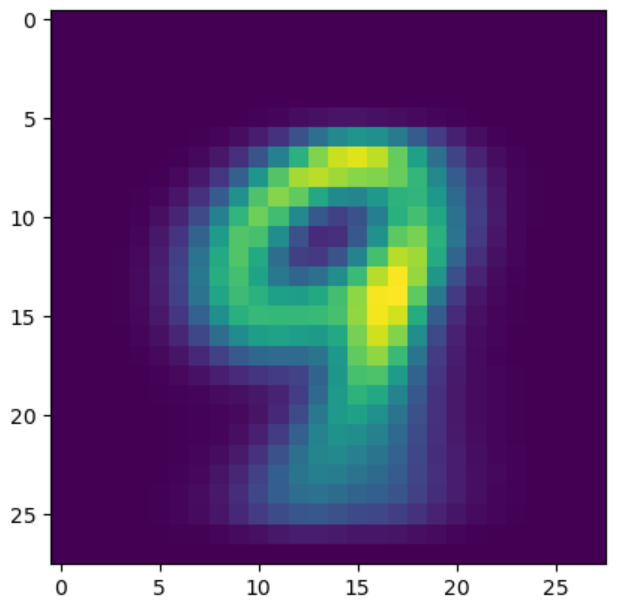
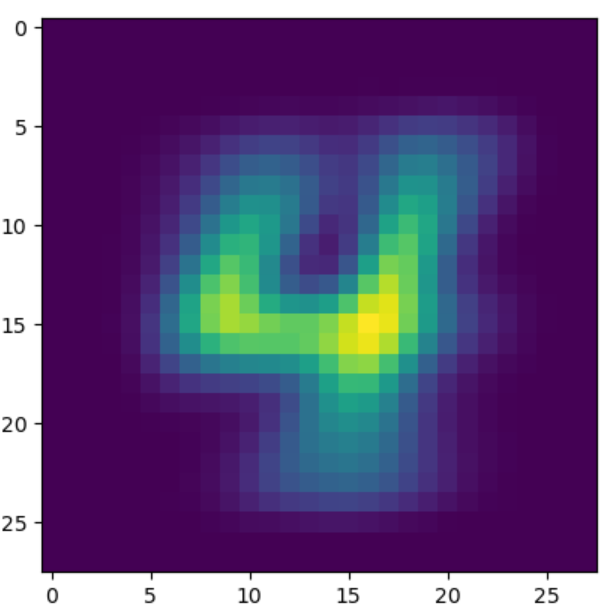


This table tells us that there are some digits the closer to each other than other digits (Measured by Euclidean distance metric). Those digits are harder for classifications then other because their similarity is higher.

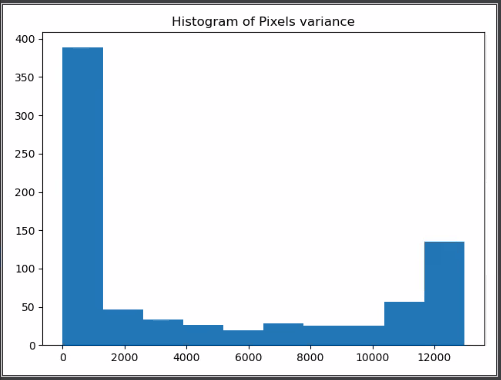
An example of a pair that is easier for classification is 1 and 0 because their distance is relatively high (1928). We can notice that their centroid looks highly different just by looking.



An example of a pair that is harder for classification is 4 and 9 because their distance is relatively low (633). Same as with 0 and 1, We can notice that their centroid looks highly similar just by looking.



**Step 4:**



**x-axis:** variance

**y-axis**: # of pixels

We computed the variance of each pixel in the entire data set.

The plot shows the # of pixels for each range of variance – with 10 equal ranges.

We can see that the left column is significantly higher than the others due to

The low variance in the color of the frame pixels and the image borders.

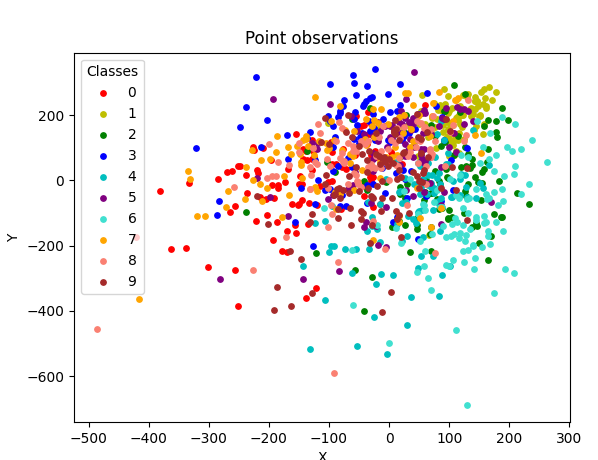
We can conclude that a lot of the pixels are of low value to us because they have a very low variance and could be removed.

**Step 5:**

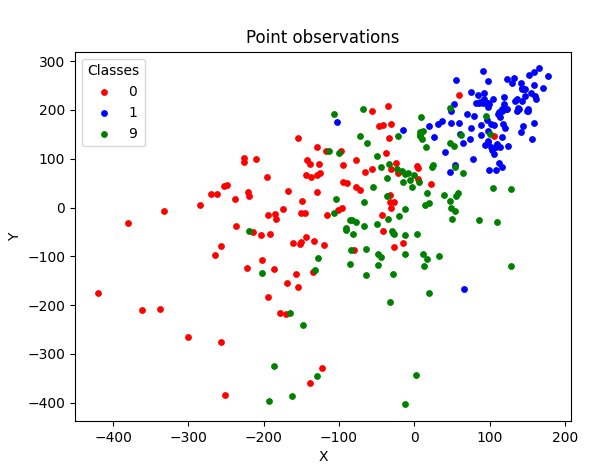
When we look at the scatter plot for all the classes, we can see that the data is not well separated, since we reduced the number of attributes from 784 to 2 and by doing that lost a lot of information.

On the scatter plot of 1,9,0, we can see that although we reduced the dimension to 2-d, the data looks more separable than the full scatter plot, which means that the PCA keep the best variances.

**Scatter plot for classes 0-9 in the 2D representation:**

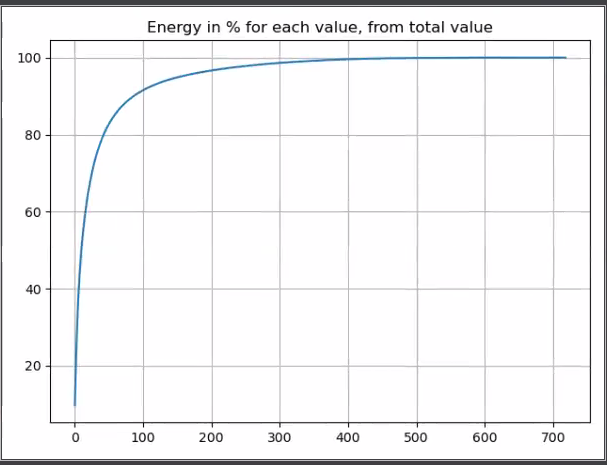


**Scatter plot for classes 0,1,9 in the 2D representation:**



**Step 6:**

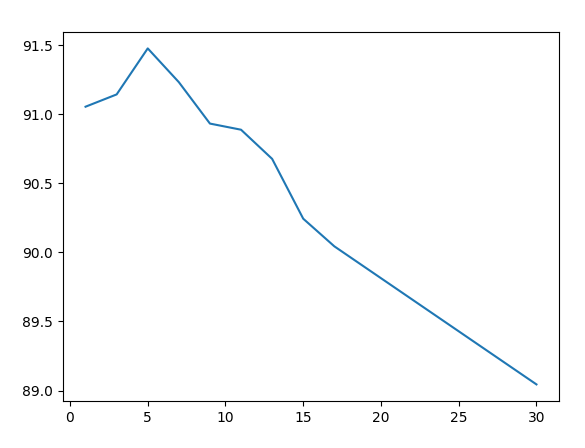
we chose a k = 87 that will retain 90% of the energy (87 dimensions).



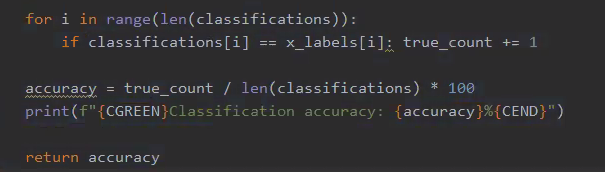
**Step 7:**

The number of dimensions will be 87 (as chosen in step 6).

The chosen k is 5 (# of neighbors), it has been selected after running accuracy test on the validation set as shown in the graph below:



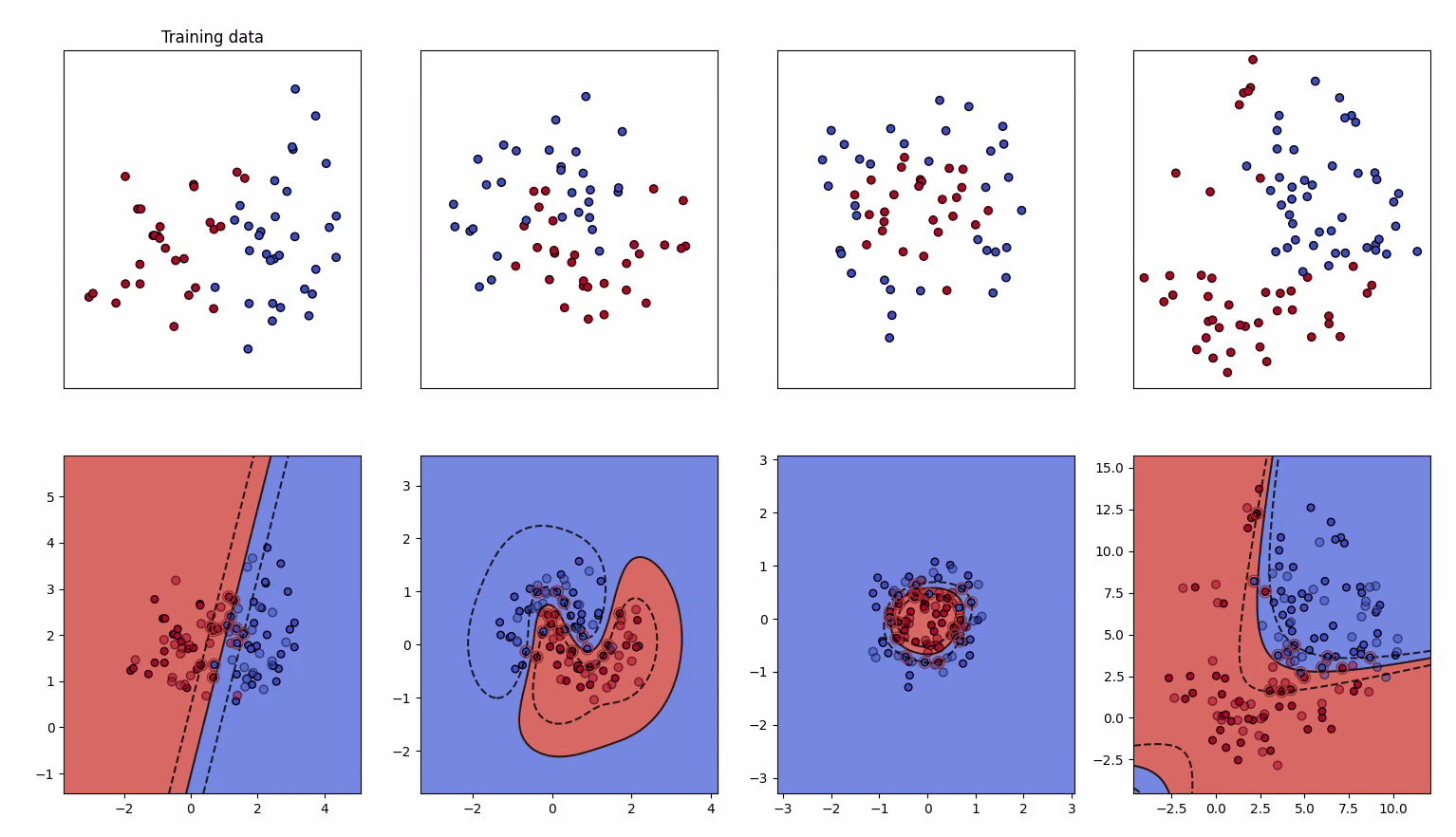
Code attached as well:



After the parameters n and k were selected, we ran KNN on the test set and got the following result:



**Question 2:**



**Dataset 1 (left graph): result = 0.95**

The data is nearly linearly separatable, so we decided to use the linear kernel with a C value of 1 (to broaden the margin and avoid overfitting).

The data is also separatable by the RBF kernel, but due to his computational costs we decided to go with the linear kernel.

**Dataset 2: result = 0.975**

Looking at the data we noticed it has a spiral shape, so we excluded the polynomial or linear kernels and decided to use the RBF kernel.

In order to avoid overfitting, we chose gamma = 1. And the C value is 6.

**Dataset 3: result = 0.95**

As we can see the red class data is bounded by the blue class data in a circle shape which calls for use in a polynomial kernel.

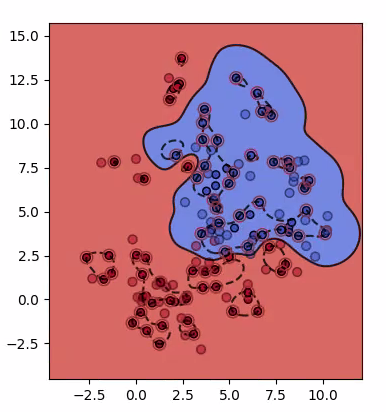
The RBF could do the job as well but we prefer to choose the least complicated kernel (which is polynomial).

**Dataset 4 (right graph) : result = 0.948**

Looking at the data, it looks like a 2nd degree polynomial kernel is the right kernel for that dataset, as it is simpler then the RBF and will

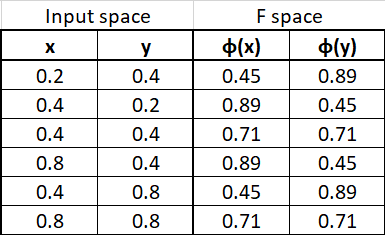
Give us a less overfitting separation. We weren’t sure so we tried using the RBF as well and the decision boundary seemed to complicated

(even though it gave a better accuracy result of 0.965):

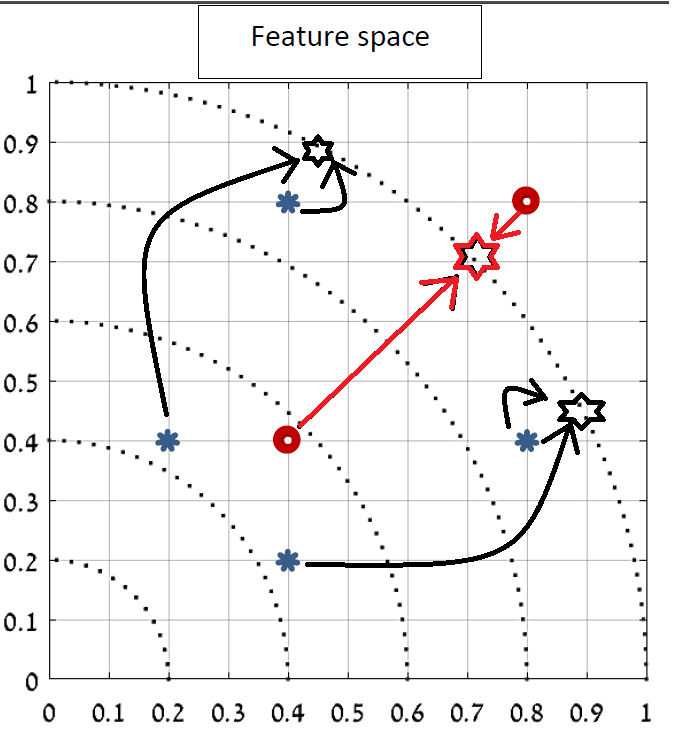


**Question 3:**

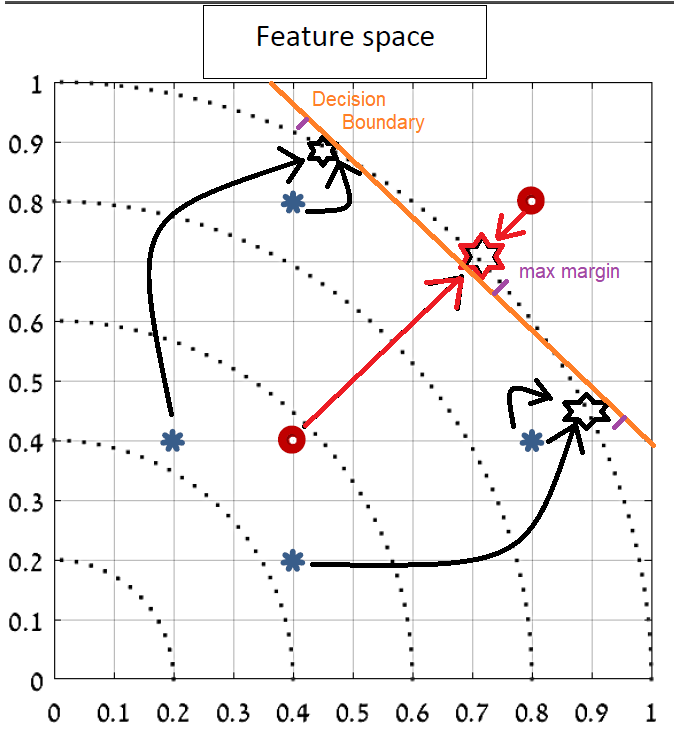
**3.1 – in excel sheet**



**3.2**



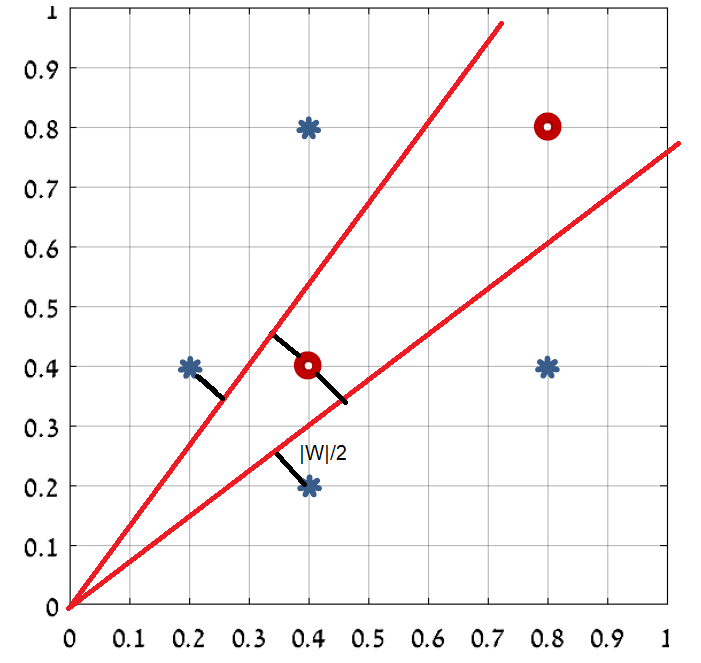
**3.3**



**3.4**

**NO** , because if we use the :

**3.5**

******