Problem 1:

Followed the steps mentioned the paper, calculated similarity using collinearity and obtained the following results:

Mean Squared Error: 0.9705905770417405
Root Mean Squared Error: 0.9851855546249856

Problem 2:

Use the SVM classifier in scikit learn and try different kernels and values of penalty param- eter. Important: Depending on your computer hardware, you may have to carefully select the parameters (see the documentation on scikit learn for details) in order to speed up the computation. Report the error rate for at least 10 parameter settings that you tried. Make sure to precisely describe the parameters used so that your results are reproducible.

SNo	Parameter Setting	Results
1.	kernel = linear C = 10	0.019499999999999962 (over-fitting)
2.	kernel = linear gamma = scale C = 0.25 tol = 1e-4	0.03420000000000001
3.	kernel = poly max_iter = 50 degree = 3	0.7501
4.	kernel = poly degree = 10 max_iter = 500	0.6844
5.	kernel = sigmoid C = 1 tol = 0.1	0.21899999999999997
6.	kernel = rbf gamma = scale class_weight = balanced max_iter = 100	0.6195999999999999
7.	kernel = sigmoid coef0 = 0.8	0.2913
8.	kernel = sigmoid coef0 = 50 max_iter = 100 class_weight = balanced	0.8865
9.	Kernel = rbf tol = 100 gamma = auto degree = 10	0.8991

10	Kernel = rbf tol = 0.001 gamma = auto degree = 10	0.051000000000000045 (overfitting)

Use the MLPClassifier in scikit learn and try different architectures, gradient descent schemes, etc. Depending on your computer hardware, you may have to carefully select the parameters of MLPClassifier in order to speed up the computation. Report the error rate for at least 10 parameters that you tried. Make sure to precisely describe the parameters used so that your results are reproducible.

SNo	Parameter Setting	Results
1.	solver: sgd alpha: default learning_rate: invscaling power_t: default shuffle: true (Default setting)	0.17369999999999997 (does not converge)
2.	learning_rate_init: 0.01 aplha: 0.001 max_iter: 100	0.026100000000000012
3.	solver: sgd alpha: 1 learning_rate_init: 0.001 learning_rate: constan tmomentum: 0.01	0.04579999999999995
4.	hidden_layer_sizes : [10,10] activation: tanh alpha: 0.1 max_iter: 500	0.055300000000000016
5.	solver: lbfgs max_iter: 1000 early_stopping: true activation: identity learning_rate_init: default	0.07769999999999999
6.	solver: lbfgs hidden_layer_sizes: [500,500] alpha: 10 early_stopping: true max_iter: 2000 activation: default	0.015900000000000025
7.	hidden_layer_sizes : [10,10] activation: tanh	0.05559999999999998

	alpha: 0.1 max_iter: 500	
8.	solver: adam activation: 'relu' alpha: 1 hidden_layer_sizes = (150,100,50,50) random_state = 1 max_iter = 750	0.03069999999999995
9.	learning_rate_init: 0.0001 alpha = 10	0.11060000000000003
10.	learning_rate_init: 10 alpha = 0.001	0.9018
11.	learning_rate_init: 0.0001 max_iter: 500 alpha: 0.001	0.11060000000000003
12.	learning_rate_init: 10 alpha: 10	0.9108

Use the k Nearest Neighbors classifier called KneighborsClassifier in scikit learn and try different parameters (see the documentation for details). Again depending on your computer hardware, you may have to carefully select the parameters in order to speed up the computation. Report the error rate for at least 10 parameters that you tried. Make sure to precisely describe the parameters used so that your results are reproducible.

Neighbors: K value
Weights: to choose between uniform KNN or weighted KNN
P: power in the Minkowski metric

SNo	Parameter Setting	Error Rate
1.	n_neighbors: 10 weights: distance p: 2	0.03159999999999996
2.	n_neighbors: 2 weights: distance p: 4 algorithm: kd_tree	0.03090000000000004
3.	n_neighbors: 3 weights: distance	0.028299999999999992
4.	n_neighbors: 6 leaf_size: 10	0.029100000000000015

	algorithm: Distance	
5.	n_neighbors: 7 weights: Uniform leaf_size: 50 algorithm: ball tree	0.03059999999999996
6.	n_neighbors: 9 weights: Uniform leaf_size: 100 algorithm: auto	0.03410000000000002
7.	n_neighbors: 1 algorithm: auto	0.03090000000000004
8.	n_neighbors: 2 weights: uniform algorithm: brute	0.0373
9.	n_neighbors: 7 Weights: distance n_jobs: 10	0.030000000000000027
10.	n_neighbors: 15 algorithm: brute Weights: distance n_jobs: 10	0.0353

What is the best error rate you were able to reach for each of the three classifiers? Note that many parameters do not affect the error rate and we will deduct points if you try them. It is your duty to read the documentation and then employ your machine learning knowledge to determine whether a particular parameter will affect the error rate. Finally, don't change just one parameter 10 times; we want to see diversity.

Classifier	Paramaters	Best Error Rate
SVM	kernel = sigmoid C = 1 tol = 0.1	0.21899999999999997
MLP	solver: lbfgs hidden_layer_sizes: [500,500] alpha: 10 early_stopping: true max_iter: 2000 activation: default	0.015900000000000025
KNN	n_neighbors: 3 weights: distance	0.028299999999999992