Machine Exercise 05/06/07 – FACIAL RECOGNITION

REPORT | Dana Kathleen Redeña

Implementation:

Programming Language: Python 3

Libraries used: scikit, numpy, scipy

- Decision Tree: tree.DecisionTreeClassifier()
- Artificial Neural Network:
  - Default: MLPClassifier(solver='sgd', activation='logistic', hidden_layer_sizes=(class_size,),max_iter=200000)
  - Varying Hidden Nodes: MLPClassifier(solver='sgd', activation='logistic', hidden_layer_sizes=(x,),max_iter=200000), *where x increments from 20 to 920 by 100's*
  - Varying Learning Rates: MLPClassifier(solver='sgd', activation='logistic', hidden_layer_sizes=(420,),max_iter=200000,alpha=lrate[x]), *where lrate is an array containing the various alpha values*
  - Varying Hidden Layers: (for 2 hidden layers) MLPClassifier(solver='sgd', activation='logistic', hidden_layer_sizes=(420,420),max_iter=200000)
- Support Vector Machines:
  - Linear Kernel: svm.SVC(kernel='linear')
  - RBF Kernel: svm.SVC(kernel='rbf')
  - Polynomial Kernel: svm.SVC(kernel='poly')
  - Sigmoid Kernel: svm.SVC(kernel='sigmoid')
  - Varying Polynomial Degrees: svm.SVC(kernel='poly', degree=x), *where x varies from 1 to 10*
- For Training: .fit(train_list, train_label)
- For Classification: .predict(test1_list) and .predict(test2_list)

Files Breakdown:

- me567.py – contains most of the algorithm (excluding only part 7 of the specs' experiments)
- scaled.py – Scales down the pixels values to only range from [0,1] and then the chosen parameters (optimal/best) for the ANN and SVM models are used
- checker.py – given the keyword for a certain filename (corresponds to a model), the program computes for the accuracy of the models
- results.txt – has the information of the results and accuracies of the experiments

Experiments:

A. **Decision Trees** *(tree.png is a visual representation of the decision tree generated)*

Test Sets	Correct	Accuracy
Test 1 (139)	124	0.8920863309352518
Test 2 (208)	196	0.9423076923076923
Total (347)	**320**	**0.9221902017**

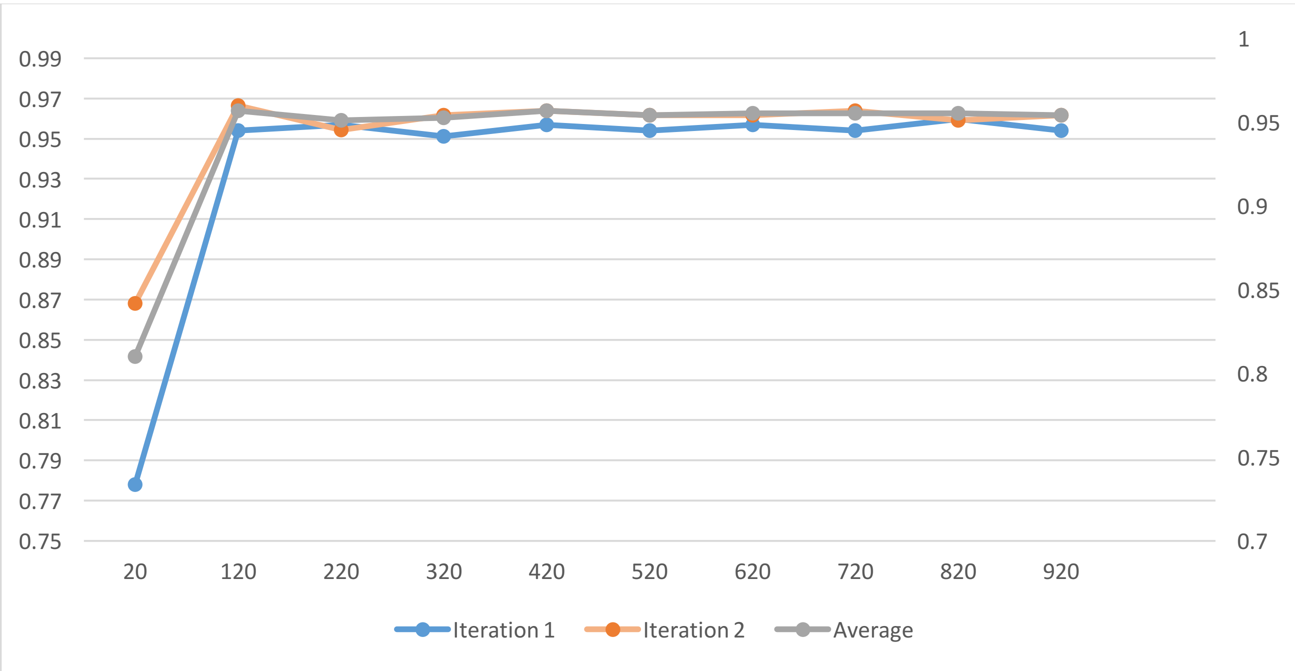
B. Artificial Neural Network

Since the Neural Networks produced change for every iteration, the results below have 2 iterations.

Default (20 hidden nodes)

Iteration	Test Sets	Correct	Accuracy
1st	**Test 1** (139)	123	0.8848920863309353
	Test 2 (208)	201	0.9663461538461539
	Total (347)	**324**	**0.9337175792**
2nd	**Test 1** (139)	124	0.8920863309352518
	Test 2 (208)	191	0.9182692307692307
	Total (347)	**315**	**0.9077809798**

Varying Hidden Nodes

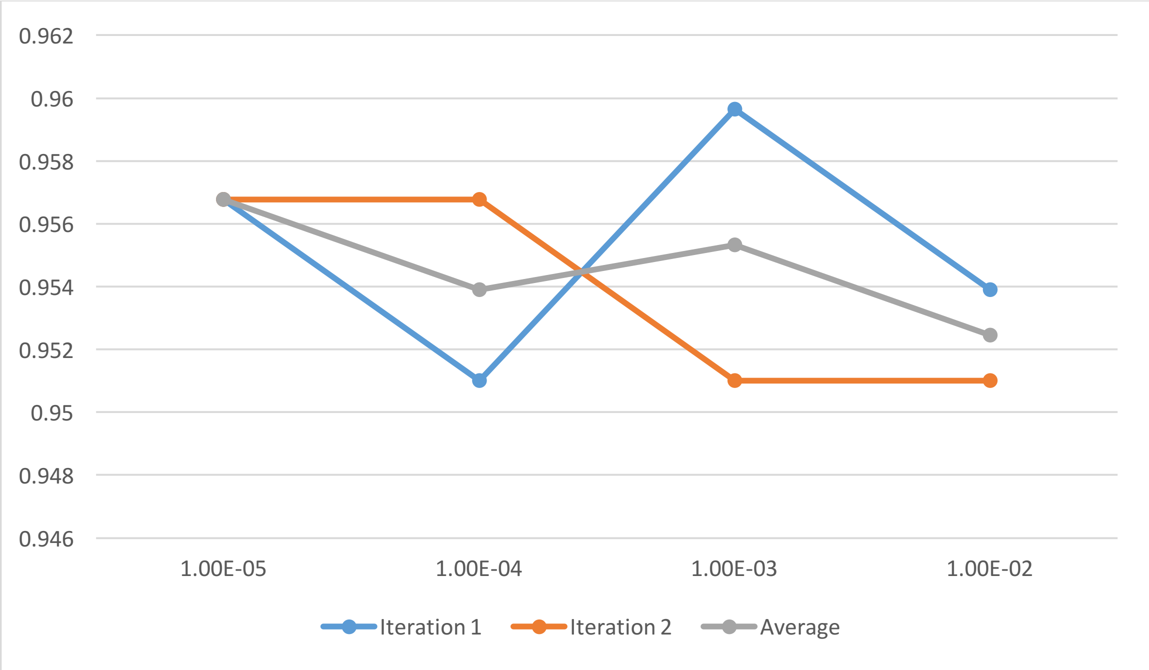


Complete breakdown in results.txt

Hidden Nodes	Iteration	Correct (out of **347**)	Accuracy
20	1st	270	0.778097982708934
	2nd	292	0.84149855907781
120	1st	331	0.953890489913545
	2nd	333	0.959654178674352
220	1st	332	0.956772334293948
	2nd	328	0.945244956772334
320	1st	330	0.951008645533141
	2nd	331	0.953890489913545
420	1st	**332**	**0.956772334293948**
	2nd	**332**	**0.956772334293948**
520	1st	331	0.953890489913545
	2nd	331	0.953890489913545
620	1st	332	0.956772334293948
	2nd	331	0.953890489913545
720	1st	331	0.953890489913545
	2nd	332	0.956772334293948
820	1st	333	0.959654178674352
	2nd	330	0.951008645533141
920	1st	331	0.953890489913545
	2nd	331	0.953890489913545

For this I've chosen that the 420-hidden node mark is the best choice because the results from its 2 iterations are equal and it has the 2^{nd} to the highest accuracy.

Varying Learning Rates

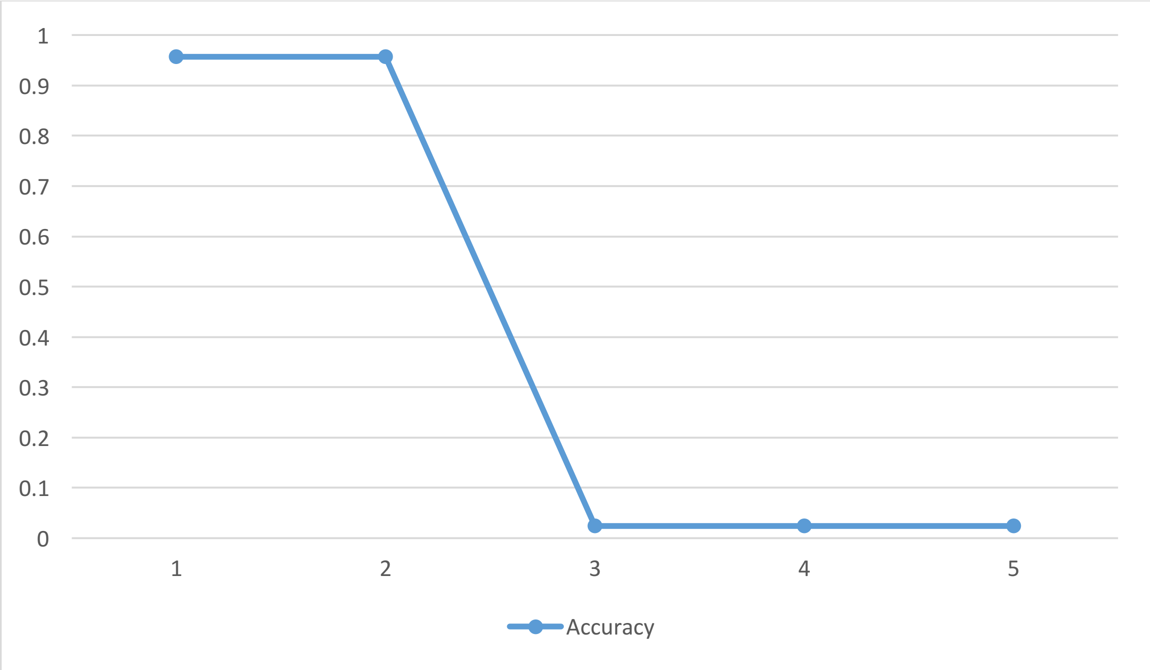


Complete breakdown in results.txt

Learning Rate	Iteration	Correct (out of **347**)	Accuracy
1e-5	1st	**332**	**0.956772334**
	2nd	**332**	**0.956772334**
1e-4	1st	330	0.951008646
	2nd	332	0.956772334
1e-3	1st	333	0.959654179
	2nd	330	0.951008646
1e-2	1st	331	0.95389049
	2nd	330	0.951008646
1e-1	1st	334	0.962536023
	2nd	333	0.959654179

For this I think the learning rate of 1e-5 is the best since it had the 2^{nd} to the highest result and there is not much discrepancy between the 2 iterations.

Varying Number of Hidden Layers



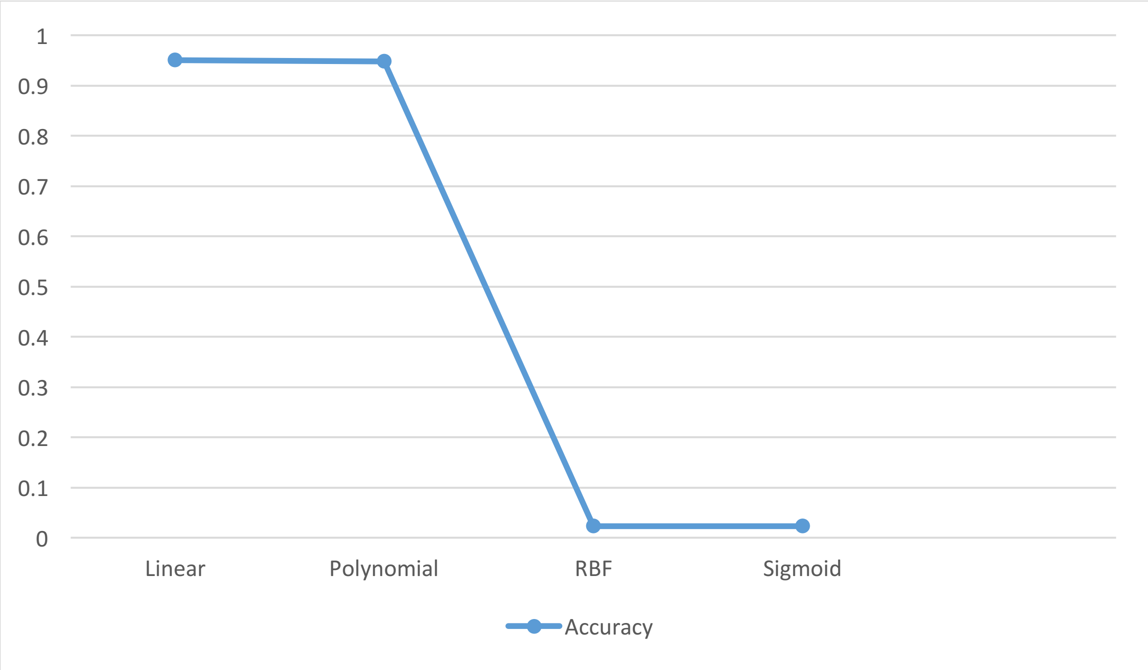
Complete breakdown in results.txt

Hidden Nodes	Correct (out of **347**)	Accuracy
1	**332**	**0.9567723342939481**
2	332	0.9567723342939481
3	8	0.023054755043227664
4	8	0.023054755043227664
5	8	0.023054755043227664

From the runs, it seems like having 3 or more hidden layers is not very effective for classification.

C. Support Vector Machines

Varying Kernels

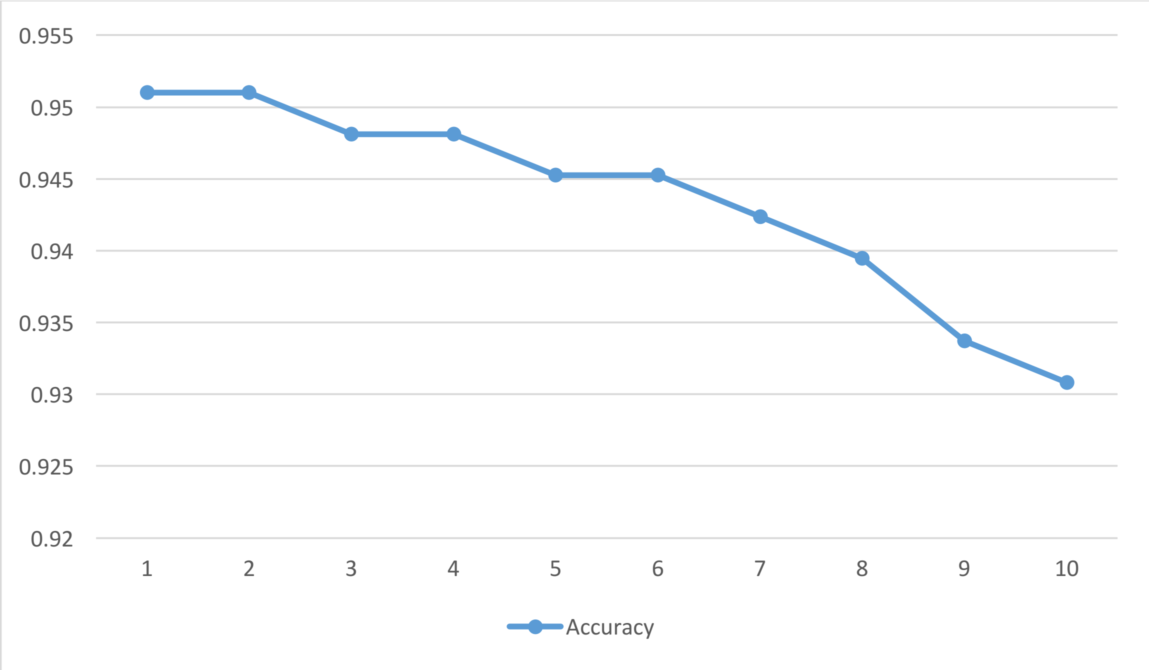


Complete breakdown in results.txt

Kernel	Correct (out of **347**)	Accuracy
Linear	**330**	**0.9510086455331412**
Polynomial	329	0.9481268011527377
RBF	8	0.023054755043227664
Sigmoid	8	0.023054755043227664

This shows that RBF and sigmoid kernels are not very good in classifying for n-classes. The linear kernel stills exceeds for this example.

Varying Polynomial Degrees



Complete breakdown in results.txt

Polynomial Degree	Correct (out of **347**)	Accuracy
1	**330**	**0.951008646**
2	330	0.951008646
3	329	0.948126801
4	329	0.948126801
5	328	0.945244957
6	328	0.945244957
7	327	0.942363112
8	326	0.939481268
9	324	0.933717579
10	323	0.930835735

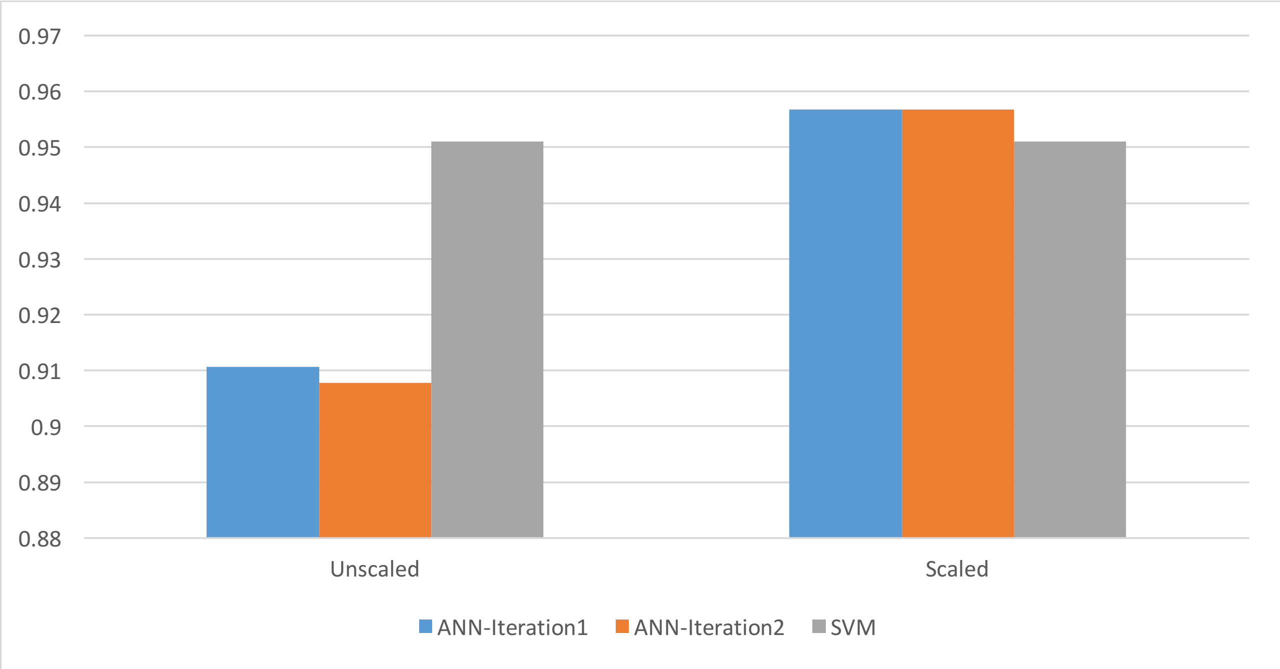
The polynomial model with degree = 1 still has the highest performance tied with degree=2 (but using degree=2 is slower than degree=1). Degree=1 also means a linear kernel (a polynomial with degree=1).

D. Scaled to [0,1]

Chosen parameters for ANN: 1 hidden layer, 420 hidden nodes, 1e-5 learning rate

Chosen parameters for SVM: kernel is linear

Training Time: training time is much slower for the scaled values



Complete breakdown in results.txt

	Model	Correct (out of **347**)	Accuracy
Scaled	ANN (1st)	332	0.9567723342939481
	ANN (2nd)	332	0.9567723342939481
	SVM	330	0.9510086455331412
Unscaled	ANN (1st)	316	0.9106628242074928
	ANN (2nd)	315	0.9077809798270894
	SVM	330	0.9510086455331412

For the ANN, the scaled values actually slow down and lessened the accuracy. For SVM, scaling down the values don't seem to make much of a difference.