Inf2B-CW2

Task 3 Report

➔ my_gaussian_classify:

- for each class of the data:
    - find the mean vector $\mu = E[x]$ from training data
    - calculate covariance matrix using $\Sigma = E[(x-\mu)(x-\mu)^T]$ and add epsilon along diagonal
    - calculate for the covariance matrix: natural logarithm determinant; inverse
    - subtract class mean vector from testing data
    - for each vector in testing data:
        - calculate (natural logarithm) posterior probability
- sort posterior probabilities according to `argmax()` to find best-matching class

Time elapsed approx (in seconds) [DICE environment, command line]	covariances	classes	total
	2.38	21.96	24.34

Statistics	N	Nerrs	acc
- initial run gives good accuracy, almost on-par with knn_classify, while only needing ~60% of the total time of knn_classify	7800	1250	83.97%

➔ my_improved_gaussian_classify:

- user-defined dimensionality $d \in [1, 26]$
- for each class of the data:
    - find eigenvalues, eigenvectors from covariance matrix
- select (dims) eigenvectors with highest eigenvalues
- apply eigenvector transformations to training and testing data
- continue with gaussian classifier on modified data as before

➔ my_improved_gaussian_system:

- parse arguments to run different preset experiments or set number of dimensions to reduce to
    - usage: `$ my_improved_gaussian_system.py [-e experiment][-d dims]`
    - choose either to override default behaviour, indicated below

Time elapsed approx (in seconds) [DICE environment, command line]	covs x26 (784x784)	eigen-vectors	PCA transf.	covs x26 (26x26)	class prob.	total
	2.39	7.05	0.41	0.02	0.47	10.34

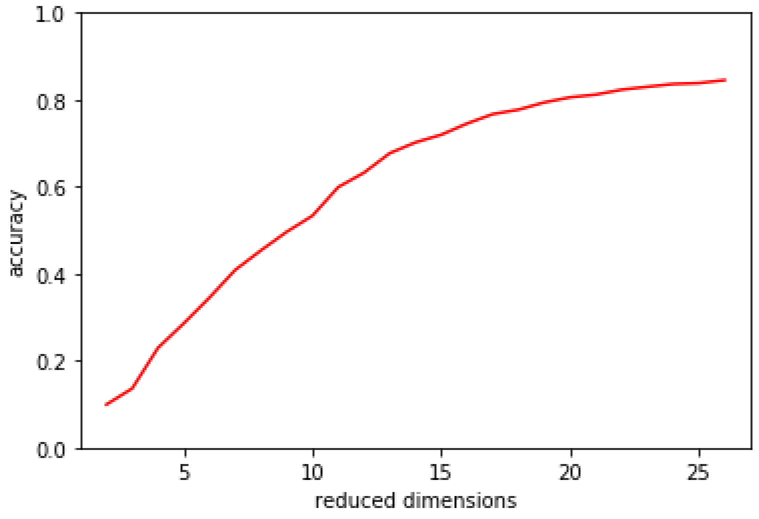
Statistics / Observations

- runs ~2x quicker than raw gaussian_classify
- accuracy slightly better with similar ε
- ε makes a difference in higher dimensions
- (ε = 0 is unsuitable, and therefore highlighted in red, as it can result in a matrix with no inverse)
- tradeoff here is ~-1% accuracy for 4x higher performance compared to knn_classify
- finding matrix eigenvectors results in complex values, that although are probably small enough to be ignored, are still present and may influence results
- takes longer real time to implement, more things could go wrong (compared to lower up-and-running time of a knn implementation)

exp.	dims	ε	N	Nerrs	acc
1	1	0.01	7800	7028	9.90%
2	1	1e-10	7800	7027	9.91%
3	2	0.01	7800	6740	13.59%
4	2	1e-10	7800	6740	13.59%
5	4	0.01	7800	5581	28.45%
6	4	1e-10	7800	5573	28.55%
7	8	1e-10	7800	3930	49.62%
8	16	1e-10	7800	1818	76.68%
9	21	1e-10	7800	1381	82.29%
10	(26)	0.02	7800	1257	83.88%
11	(26)	0.01	7800	1216	84.40%
12	(26)	1e-10	7800	1167	85.03%
13	(26)	0	7800	1167	85.03%

Legend:

▨ [point of comparison (ε=0.01)]
⬜ [ε altered]
🟩 [best PCA run (ε=0.01)]
🟩 [best PCA result with modded ε=1e-10]
🟥 [best theoretical result, ε=0, risky]



plot of reduced PCA dimensions vs accuracy (with ε=1e-10)