INF8245E: Machine Learning | Assignment #2

Louis Plessis (1933334)

24 October 2021

Acknowledgement: I have briefly discussed about this assignment with Marie-Christine Paré and Joseph Battesti

1. Linear Classification and Nearest Neighbor Classification

1.1.

The dataset can be found in “DS1_test.txt”, “DS1_train.txt”, and “DS1_valid.txt”.

1.2. GDA model *(results can vary since the 2000 examples are generated randomly)*

1.2.1. Best fit accuracy

Best fit accuracy achieved by the classifier: 0.95875

1.2.2. Learnt coefficients

Mean_negative = [1.38796111 1.38480463 1.34313205 1.32022768 1.41542407 1.31962424 1.38474112 1.31388446 1.44811996 1.38415931 1.43311451 1.37802054 1.39557523 1.47770518 1.42522795 1.34860382 1.40656817 1.35347971 1.46513806 1.38616681]

Mean_positive = [1.77001928 1.83872663 1.80206517 1.87441891 1.80576205 1.82425626 1.84645289 1.87175208 1.89023374 1.84551547 1.88524332 1.88144806 1.8342172 1.82871093 1.87524358 1.90254666 1.8618322 1.8550372 1.85403167 1.87095304]

Covariance = (see Jupyter Notebook)

P_negative = 0.5041666666666667

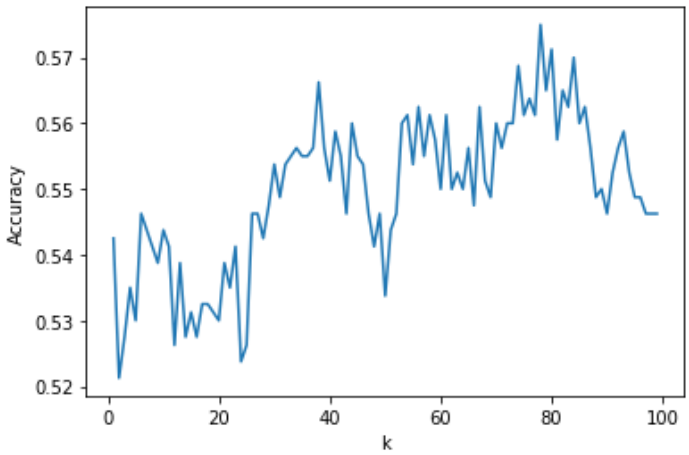
P_positive = 0.49583333333333335

w = [13.88253986 -8.22904992 -5.83624115 -3.4228207 -9.30239267 -4.00878667 16.83588069 -22.92527894 -28.30746076 8.57953273 -12.57712048 -12.35077779 15.39786132 12.79502454 -5.23417678 12.41190246 28.67152986 -6.43891018 -1.05272077 -4.84561611]

w0 = 26.5519939814707

Confusion matrix: [[388, 14], [19, 379]]

1.3. K-NN *(results can vary since the 2000 examples are generated randomly)*



1.3.1. This classifier seems to perform worse than GDA overall (~0.55 accuracy instead of ~0.95 for GDA). Some specific values of k seem to perform better than others, due to the low stability of this method. However, the difference is very small as the accuracy stays around 55%.

1.3.2. Best fit accuracy

Best fit accuracy: 0.575 when k = 78

1.4. Mixture of 3 Gaussians

The dataset can be found in "DS2_test.txt", "DS2_train.txt", and "DS2_valid.txt".

1.5. DS2 *(results can vary since the 2000 examples are generated randomly)*

1.5.1. GDA model

1.5.1.1. Best fit accuracy

Best fit accuracy achieved by the classifier: 0.5075

1.5.1.2. Learnt coefficients

Mean_negative = [1.27256109 1.24218875 1.2914599 1.33057664 1.29511144 1.18556019 1.24628351 1.25635201 1.26262108 1.22139844 1.31234446 1.33184905 1.29710941 1.15561398 1.26534078 1.26492537 1.32194299 1.32341061 1.24851654 1.29086104]

Mean_positive = [0.92068364 0.96828546 0.98662197 0.93430012 0.96447765 1.00666499 0.94414745 0.96325523 0.96284449 0.97270541 0.91646454 0.95894061 1.02767194 0.96403723 0.92911611 0.99505608 1.03159224 1.01400484 1.0095793 1.00980303]

Covariance = (see Jupyter Notebook)

P_negative = 0.5104166666666666

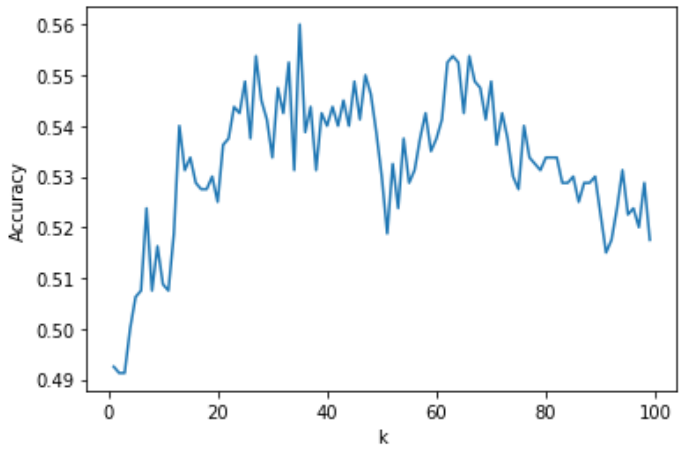
P_positive = 0.4895833333333333

w = [0.03814785 0.0026456 -0.03497897 0.10577675 0.10759092 -0.06281204 -0.02189556 -0.02961159 0.05787611 -0.01387161 0.06280913 0.04418121 0.00409997 -0.051621   -0.01852591 -0.01495493 0.00869919 -0.03507169 -0.03039875 -0.01882013]

w0 = -0.07478167001750505

Confusion matrix: [[191, 170], [224, 215]]

1.5.2. K-NN



This classifier seems to perform a little bit better than GDA (~0.55 accuracy instead of ~0.50 for GDA). Some specific values of k seem to perform better than others, due to the low stability of this method. However, the difference is very small as the accuracy stays around 55%.

1.5.3. Best fit accuracy with K-NN

Best fit accuracy: 0.56 when k = 35

1.6. Similarities and differences between the performance of both classifiers

For DS1, GDA seems to perform better than k-NN, but for DS2, it seems to be the opposite. However, k-NN behaves in the same way with both datasets (i.e., some values of k perform better than others, with no visible pattern). Given the very high accuracy for the GDA model using DS1 and the average performance of both models with DS2, we could say that GDA performs better overall.

2. MNIST Handwritten Digits Classification

2.1. GNB model

Data preprocessing steps can be found in the Jupyter Notebook.

2.1.1. Equations for mean and diagonal covariance matrices

Prior class probability for each class i:

$$P = \frac{\text{number of examples of class } i}{\text{total number of examples}}$$

Mean for each class i:

$$\mu_i = \frac{1}{N_i} \sum_{n=1}^{N} t_i^{(n)} x^{(n)}$$

Covariance matrix for each class i:

$$\Sigma_i = \frac{1}{N_i} \sum_{n=1}^{N} t_i^{(n)} (x^{(n)} - \mu_i)(x^{(n)} - \mu_i)^T$$

Since the covariance matrix must be diagonal, each value that is not on the diagonal is 0.

2.1.2. Estimating GNB model parameters

********** Class 0 **********

P = 0.09864

Mean, Covariance: See Jupyter notebook

********** Class 1 **********

P = 0.11356

Mean, Covariance: See Jupyter notebook

********** Class 2 **********

P = 0.09936

Mean, Covariance: See Jupyter notebook

********** Class 3 **********

P = 0.10202

Mean, Covariance: See Jupyter notebook

********** Class 4 **********

P = 0.09718

Mean, Covariance: See Jupyter notebook

********** Class 5 **********

P = 0.09012

Mean, Covariance: See Jupyter notebook

********** Class 6 **********

P = 0.09902

Mean, Covariance: See Jupyter notebook

********** Class 7 **********

P = 0.1035

Mean, Covariance: See Jupyter notebook

********** Class 8 **********

P = 0.09684

Mean, Covariance: See Jupyter notebook

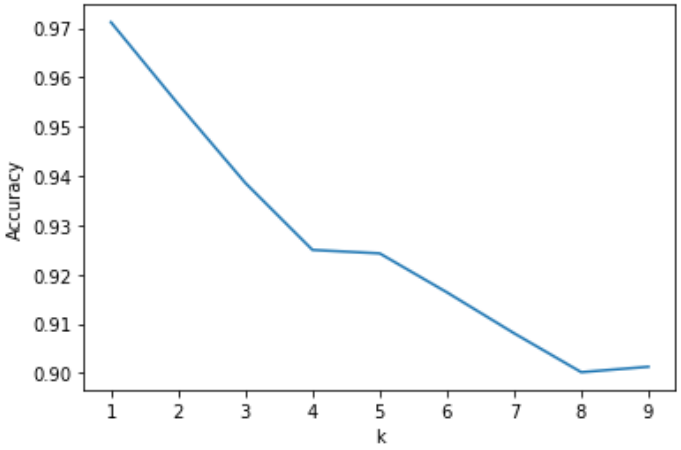
********** Class 9 **********

P = 0.09976

Mean, Covariance: See Jupyter notebook

Unfortunately, I was not able to compute predictions, since the covariance matrices could not be inverted (Laplace smoothing doesn't seem to work properly) despite spending more than 10 hours trying to solve the problem. Hence, the best fit accuracy could not be found, but the methods have been implemented in the Jupyter Notebook.

2.2. K-NN



2.2.1. It seems like k=1 performs best and that the accuracy drops as the value of k increases. This is probably because we have 10 classes in this case, and therefore that the class distribution is not locally smooth.

2.2.2. Best fit accuracy: 0.9712 when k = 1

2.3. GNB performance vs k-NN

Since I could not compute the performance of GNB, I am unable to compare it with k-NN. However, given the very high accuracy found with k-NN, we can expect that the performances will be either very similar or that k-NN performs better (for a very low value of k only). My prediction would be that GNB would not perform is good as k-NN, since we are using Laplace smoothing for the covariance matrix and therefore adding noise to the data.