COMP 3105 – Assignment 3 Report
– Fall 2025 –

Due: Sunday November 16, 2025 23:59.
Group 51
Andrew Wallace - 101210291
Christer Henrysson - 101260693

Getting started
Note that Python 3.11 is used for this assignment. Please install requirements using virtual environment via:

```
python3.11 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Question 1 (4%) Linear Multi-Class Classifier

(a) (1%) Implement a Python function
`W = minMulDev(X, Y)`
Recall that multinomial deviance loss is given by:

$$W^* = \text{argmin}_W \in \mathbb{R}^{d\times k}\frac{1}{n}\sum_{i=1}^{n}\log(1_k^{\mathrm{T}}\exp^{W^T x_i}) - y_i^{\mathrm{T}}W^{\mathrm{T}}x_i$$

Please see `A3codes.py` for the implementation of `W = minMulDev(X, Y)`

(b) (1%) Implement a Python function
`Yhat = classify(Xtest, W)`
Please see `A3codes.py` for the implementation of `Yhat = classify(Xtest, W)`

(c) (1%) Implement a Python function
`acc = calculateAcc(Yhat, Y)`
Please see `A3codes.py` for the implementation of `acc = calculateAcc(Yhat, Y)`

(d) (1%) In this part, you will evaluate your implementation on the synthetic datasets from above. The results from the synthetic classification experiment using seed 51 report the following training accuracies:

n	Model 1	Model 2
16	0.93125	1
32	0.88125	0.984375
64	0.86875	0.9609375
128	0.85390625	0.94296875

Table 1: Training accuracies with different number of training dataset sizes

The results from the synthetic classification experiment using seed 51 report the following test accuracies:

n	Model 1	Model 2
16	0.7317	0.8963
32	0.7928	0.8857
64	0.8228	0.8976
128	0.844	0.9178

Table 2: Test accuracies with different number of training dataset sizes

Our results show that as the size of the input dataset increases (n) the accuracy decreases in both models. This is because the classifier can overfit the small dataset size. As the size of the dataset increases, there are more overlapping data points which decreases the accuracy of the classifier. We see this impacting Model 1 more than Model 2 since the data in Model 2 is more linearly separable compared to Model 1.

We see the opposite effect in the test dataset. This is because the classifier becomes more generalized (less overfitting) when the training points increase. So, we see the more generalized classifier performing better to unseen datasets (the test dataset). Furthermore, the classification is less accurate on smaller dataset sizes since the classifier becomes overfitted to the training dataset. Again, we see that the accuracies are higher for Model 2 since it is more linearly separable compared to Model 1.

Question 2 (7%) Principle Component Analysis

(a) (1%) Implement a Python function
`U = PCA(X, k)`
Please see `A3codes.py` for the implementation of `PCA(X, k)`

(b) (0.5%) Implement a Python function
`Xproj = projPCA(Xtest, mu, U)`
Please see `A3codes.py` for the implementation of `projPCA(Xtest, mu, U)`

(c) (2%) Implement a Python function
`A = kernelPCA(X, k, kernel func)`
Please see `A3codes.py` for the implementation of `kernelPCA(X, k, kernel func)`

(d) (2%) Implement a Python function
`Xproj = projKernelPCA(Xtest, Xtrain, kernel func, A)`
Please see `A3codes.py` for the implementation of `projKernelPCA(Xtest, Xtrain, kernel func, A)`

(e) (1%) In this part, you will evaluate your implementation on the synthetic datasets from above. Implement a Python function
`train_acc, test_acc = synClsExperimentsPCA()`
Please see `A3codes.py` for the implementation of `synClsExperimentsPCA()`
The results from the synthetic PCA classification experiment using seed 51 report the following training accuracies:

Dim k	Model 1	Model 2
1	0.8503125	0.64242188
2	0.85789063	0.94179687

Table 3: Training accuracies for PCA classification with different dimension sizes

The results from the synthetic PCA classification experiment using seed 51 report the following test accuracies:

Dim k	Model 1	Model 2
1	0.84541	0.63417
2	0.83993	0.91801

Table 4: Test accuracies for PCA classification with different dimension sizes

(f) (0.5%) Looking at your tables from above, analyze the results and discuss any findings you may have and the possible reason behind them.

We see Model 1 has similar accuracies in both the training and test data, regardless of dimension size. This is because the structure of model 1 is linear. So, when projected onto a single dimension (line), the class structure is mostly preserved and has minimal class overlap. This leads to a good accuracy when we do our prediction with multinomial deviance loss.

Interestingly, we see similar accuracies when introducing a second dimension. This is because the second dimension does not have a high variance and does little in maintaining class structure in the the projection. That is, most of the variance lies in the first dimension PCA, so adding another dimension does not increase the accuracy by much.

However, in Model 2 we see the accuracy increase when we introduce a second dimension (from dimension $k = 1$ to dimension $k = 2$). This is because the classes of data in Model 2 is clustered into different quadrants. So, when we project onto a single dimension, there will be overlap between the classes, leading to poor predication accuracy. When we add the second dimension, the class structure is preserved and leads to a higher prediction accuracy. We see this in both the training and test data.

Question 3 (4%) k-means

(a) (1%) Implement a Python function
`Y, U, obj_val = kmeans(X, k, max iter=1000)`
Please see `A3codes.py` for the implementation of `kmeans(X, k, max iter=1000)`

(b) (1%) Implement a Python function
`Y, U, obj_val = repeatKmeans(X, k, n runs=100)`
Please see `A3codes.py` for the implementation of `repeatKmeans(X, k, n runs=100)`

(c) (1%) Implement a Python function
`obj_val, list = chooseK(X, k candidates=[2,3,4,5,6,7,8,9])`
Please see `A3codes.py` for the implementation of `chooseK(X, k candidates=[2,3,4,5,6,7,8,9])`

k	2	3	4	5	6	7	8	9
`obj_val`	2.7363	1.6877	1.0105	0.8407	0.7095	0.5935	0.5062	0.4314

Table 5: Objective values for different k

Based on the list of objective values, it seems like the largest drop happens from $k = 2$ and $k = 4$. Everything after that has minimal returns. Therefore, the best k is $k = 4$.

(d) (2%) Implement a Python function
`Y, obj_val = kernelKmeans(X, kernel_func, k, init_Y, max_iter=1000)`
Please see `A3codes.py` for the implementation of `kernelKmeans(X, kernel_func, k, init_Y, max_iter=1000)`