# COMP 3105 – Assignment 2 Report
# – Fall 2025 –

**Due:** Sunday October 19, 2025 23:59.

Group 51
Andrew Wallace - 101210291
Christer Henrysson - 101260693

**Getting started**
Note that Python 3.10 is used for this assignment. Please install requirements using virtual environment via:

```
python3.10 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

## Question 1 (5%) Binary Classifier (Primal Form)

In this question, you will implement binary classification with different losses from scratch, in Python using NumPy/SciPy, and evaluate their performances on the synthetic datasets from above with different regularization hyper-parameters. You will learn some essential built-in functions like `scipy.optimize.minimize` to solve unconstrained problems and `cvxopt.solvers.qp` to solve quadratic programmings. The input vectors are assumed to be **un-augmented** in this question (i.e. we do not add a constant feature of 1 to it). All of the following functions must be able to handle arbitrary $n > 0$ and $d > 0$. The vectors and matrices are represented as NumPy arrays. Your functions shouldn't print additional information to the standard output.

(a) (1%) Implement a Python function

```
w, w0 = minExpLinear(X, y, lamb)
```

Please see `A2codes.py` for the implementation of `minExpLinear(X, y, lamb)`

(b) (1%) Implement a Python function

```
w, w0 = minHinge(X, y, lamb, stablizer=1e-5)
```

Please see `A2codes.py` for the implementation of `minHinge(X, y, lamb, stabilizer=1e-5)`

(c) (1%) Implement a Python function

```
yhat = classify(Xtest, w, w0)
```

Please see `A2codes.py` for the implementation of `classify(Xtest, w, w0)`

(d) (1%) In this part, you will evaluate your implementation with different regularization hyper-parameters. Implement a Python function

```
train_acc, test_acc = synExperimentsRegularize()
```

Please see `A2codes.py` for the implementation of `synExperimentsRegularize()`
The averages over 100 runs for each accuracy (one for training and the other for test) can be seen in the following tables.

| $\lambda$ | ExpLinear | | | Hinge | | |
|---|---|---|---|---|---|---|
| | Model 1 | Model 2 | Model 3 | Model 1 | Model 2 | Model 3 |
| 0.001 | 0.9999 | 0.5969 | 0.8768 | 0.999 | 0.6162 | 0.8711 |
| 0.01 | 0.998 | 0.6032 | 0.8761 | 0.995 | 0.6149 | 0.8698 |
| 0.1 | 0.9962 | 0.5935 | 0.8743 | 0.9909 | 0.6105 | 0.8692 |
| 1.0 | 0.9882 | 0.5981 | 0.8681 | 0.9816 | 0.6136 | 0.8644 |

Table 1: Training accuracies with different hyper-parameters

| $\lambda$ | ExpLinear | | | Hinge | | |
|---|---|---|---|---|---|---|
| | Model 1 | Model 2 | Model 3 | Model 1 | Model 2 | Model 3 |
| 0.001 | 0.98979 | 0.5655 | 0.88019 | 0.98812 | 0.60035 | 0.87802 |
| 0.01 | 0.98967 | 0.58031 | 0.88161 | 0.98608 | 0.60598 | 0.87887 |
| 0.1 | 0.98628 | 0.57143 | 0.88118 | 0.98253 | 0.60263 | 0.8773 |
| 1.0 | 0.98098 | 0.57876 | 0.87376 | 0.97381 | 0.60573 | 0.87105 |

Table 2: Test accuracies with different hyper-parameters

(e) (1%) Looking at your tables from above, analyze the results and discuss any findings you may have and the possible reasons behind them.
We can see that the training accuracies are higher when $\lambda$ approaches zero. This is because when lambda approaches zero, the regularization term $\frac{1}{2}\lambda||\mathbf{w}||_2^2$ also approaches zero. This term is used to prevent overfitting by penalizing complex models. That is, the greater $||\mathbf{w}||_2^2$ is (more complex), the greater the value of our objective function will be. The parameter $\lambda$ determines how complex our model is allowed to be. When $\lambda$ is large, the regularization term dominates, and our objective turns into minimizing the complexity of our model. When $\lambda$ is small, the loss function dominates, and our objective turns into minimizing the loss of our model. So, we see the accuracies of our training data increase (from model to model) as our $\lambda$ decreases, since we primarily focus on minimizing the loss. Conversely, as $\lambda$ gets larger (closer to 1), our accuracies decrease.

Note that this is not necessarily true when it comes to testing. If our model is overfitted to our training data, then our accuracies for test data will suffer. See the test accuracy for `ExpLinear`, Model 2. Here we can see that $\lambda = 0.01, 0.1, \text{and} 1.0$ all have higher accuracies compared to $\lambda = 0.001$. This is likely due to the simplified models used (as $\lambda$ increases) being a better fit for the test data.

## Question 2 (5%) Binary Classification (Adjoint Form)

In this question, you will implement binary classification with different losses (again) using the adjoint formula coming from the representer theorem, and evaluate their performances on the synthetic datasets from above with different kernels. The input vectors are assumed to be **un-augmented** in this question (i.e. we do not add a constant feature of 1 to it). All of the following functions must be able to handle arbitrary $n > 0$ and $d > 0$. The vectors and matrices are represented as NumPy arrays. Your functions shouldn't print additional information to the standard output.

(a) (1%) Implement a Python function

```
a, a0 = adjExpLinear(X, y, lamb, kernel_func)
```

(b) (1%) Implement a Python function

```
a, a0 = adjHinge(X, y, lamb, kernel_func, stabilizer=1e-5)
```

(c) (1%) Implement a Python function

```
yhat = adjClassify(Xtest, a, a0, X, kernel_func)
```

(d) (1%) In this part, you will evaluate your implementation with different kernels. Implement a Python function

```
train_acc, test_acc = synExperimentsKernel()
```

(e) (1%) Looking at your tables from above, analyze the results and discuss any findings you may have and the possible reasons behind them.

## Question 3 (5%) Binary Classification (SVM Dual Form)

In this question, you will implement binary classification with the hinge loss (yet again) using the dual formula, and choose the best hyper-parameter and kernel for some real-world problems via cross-validation. The input vectors are assumed to be **un-augmented** in this question (i.e. we do not add a constant feature of 1 to it). All of the following functions must be able to handle arbitrary $n > 0$ and $d > 0$. The vectors and matrices are represented as NumPy arrays. Your functions shouldn't print additional information to the standard output.

(a) (1%) Implement a Python function

```
a, b = dualHinge(X, y, lamb, kernel_func, stabilizer=1e-5)
```

(b) (1%) Implement a Python function

```
yhat = dualClassify(Xtest, a, b, X, y, lamb, kernel_func)
```

(c) (2%) The A2files.zip includes an image dataset, A2train.csv, of handwritten digits taken from the MNIST dataset. Each image is either digit 4 or digit 9 (once loaded, you can call the `plotDigit` function to see some samples of the images as in Fig. 4). The first column of the csv file is the class label. Treat digit 4 as the -1 class and digit 9 as the +1 class, your task is to use your `dualHinge` function to learn a good binary classifier. In this part, you need to perform cross-validation and select the best hyperparameters and kernels for this dataset. Implement a Python function

```
cv acc, best lamb, best kernel = cvMnist(dataset_folder, lamb_list,
                          kernel_list, k=5)
```

(d) (1%) We will evaluate your choices from (c) on a new test dataset (that you don't have access to). You will get full mark here if your chosen hyper-parameters, kernel function, dualHinge and dualClassify can achieve acceptable performance on the test dataset.