COMP 3105 – Assignment 1 Report
– Fall 2025 –

Due: Sunday September 28, 2025 23:59.

Group 51
Andrew Wallace - 101210291
TODO: Add name and student number

Question 1: (7.5%) Linear Regression

(a) **(1%)** L_2 **Regression**
Please see A1codes.py for implementation.

(b) (3%) L_∞ **Regression**
Here we are going to solve the L_∞ loss regression problem

$$\mathbf{w} = \text{argmin}_{\mathbf{w}\in\mathbb{R}^d} = ||X\mathbf{w} - \mathbf{y}||$$

Recall that this optimization can be expressed as a linear programming problem with the joint paramters $\begin{bmatrix} \underset{d\times 1}{\mathbf{w}} \\ \underset{1\times 1}{\delta} \end{bmatrix} \in \mathbb{R}^{d+1}$ as follows

$$\begin{gathered} \min_{\mathbf{w},\delta}\delta \\ \text{s.t.} \\ \delta \geq 0 \iff -\delta \leq 0 \\ X\mathbf{w} - \mathbf{y} \preceq \delta\cdot\mathbf{1}_n \iff X\mathbf{w} - \delta\cdot\mathbf{1}_n \preceq \mathbf{y} \\ \mathbf{y} - X\mathbf{w} \preceq \delta\cdot\mathbf{1}_n \iff -X\mathbf{w} - \delta\cdot\mathbf{1}_n \preceq -\mathbf{y} \end{gathered}$$

In the following answers, we will convert the optimization to a form that is solvable by the **cxvopt** linear programming (LP) solver, which solves the following form of LP

$$\begin{gathered} \min_{\mathbf{u}}\mathbf{c}^T\mathbf{u} \\ \text{s.t. } G\mathbf{u} \preceq \mathbf{h} \end{gathered}$$

Let the unknown variables be $\mathbf{u} = \begin{bmatrix} \underset{d\times 1}{\mathbf{w}} \\ \underset{1\times 1}{\delta} \end{bmatrix} \in \mathbb{R}^{d+1}$

For the constraints, since we have three sets of constraints, the matrix G and $\mathbf{h}$ can be decomposed into three parts

$$G\cdot\mathbf{u} = \begin{bmatrix} \underset{1\times(d+1)}{G^{(1)}} \\ \underset{n\times(d+1)}{G^{(2)}} \\ \underset{n\times(d+1)}{G^{(3)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix} \preceq \begin{bmatrix} \underset{1\times 1}{\mathbf{h}^{(1)}} \\ \underset{d\times 1}{\mathbf{h}^{(2)}} \\ \underset{d\times 1}{\mathbf{h}^{(3)}} \end{bmatrix}$$

(b.1) (0.25%) For the objective function, we want $\mathbf{c}^T\mathbf{u} = \delta$. What should $\mathbf{c} \in \mathbb{R}^{d+1}$ be?
Recall that $\mathbf{u} = \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix}$
For the objective function to be δ we have:

$$\mathbf{c}^T\mathbf{u} = [c_1, c_2, \dots c_d, c_{d+1}] \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ \delta \end{bmatrix} = c_1 w_1 + c_2 w_2 + \dots + w_d u_d + c_{d+1}\delta$$

Now let $c_1 = c_2 = \dots = c_d = 0$
And $c_{d+1} = 1$
This gives us:
$0w_1 + 0w_2 + \dots 0w_d + \delta$
$= \delta$.

Thus $\mathbf{c} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$

Where $c_1 = c_2 = \dots = c_d = 0$ and $c_{d+1} = 1$.

(b.2) (0.25%) We want $G^{(1)}\mathbf{u} \preceq \mathbf{h}^{(1)} \iff \delta \geq 0$. What should $G^{(1)} \in \mathbb{R}^{1\times(d+1)}$ and $h^{(1)} \in \mathbb{R}$ be?
Recall the first constraint: $-\delta \leq 0$
We want

$$\underset{1\times(d+1)}{G^{(1)}} \cdot \mathbf{u} \preceq \underset{1\times 1}{\mathbf{h}^{(1)}}$$

We will now map our constraint into this form.

$$\begin{bmatrix} \underset{1\times d}{G^{(11)}} & \underset{1\times 1}{G^{(12)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix} \preceq \underset{n\times 1}{\mathbf{h}^{(1)}} \iff -\delta \leq 0$$
$$\underset{1\times d}{G^{(11)}} \cdot \mathbf{w} + \underset{1\times 1}{G^{(12)}} \cdot \delta = -\delta \leq 0 = \underset{1\times 1}{\mathbf{h}^{(1)}}$$

So,
$\underset{1\times d}{G^{(11)}} = \mathbf{0}_{1\times d}$
$\underset{1\times 1}{G^{(12)}} = -1$
$\underset{1\times(d+1)}{G^{(1)}} = \begin{bmatrix} \mathbf{0}_{1\times d} & -1 \end{bmatrix}$
$\underset{1\times 1}{\mathbf{h}^{(1)}} = \mathbf{0}_{1\times 1}$

(b.3) (0.25%) We want $G^{(2)}\mathbf{u} \preceq \mathbf{h}^{(2)} \iff X\mathbf{w} - \mathbf{y} \preceq \delta \cdot \mathbf{1_n}$. What should $G^{(2)} \in \mathbb{R}^{n\times(d+1)}$ and $h^{(2)} \in \mathbb{R}^n$ be?
Recall our second constraint is
$X\mathbf{w} - \mathbf{y} \preceq \delta \cdot \mathbf{1}_n \iff X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq \mathbf{y}$
We want

$$\underset{n\times(d+1)}{G^{(2)}} \cdot \mathbf{u} \preceq \underset{n\times 1}{\mathbf{h}^{(2)}}$$

We will now map our constraint into this form.

$$\begin{bmatrix} \underset{n\times d}{G^{(21)}} & \underset{n\times 1}{G^{(22)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix} \preceq \underset{n\times 1}{\mathbf{h}^{(2)}} \iff X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq \mathbf{y}$$
$$\underset{n\times d}{G^{(21)}} \cdot \mathbf{w} + \underset{n\times 1}{G^{(22)}} \cdot \delta = X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq \mathbf{y} = \underset{n\times 1}{\mathbf{h}^{(1)}}$$

So,
$\underset{n\times d}{G^{(21)}} = X$
$\underset{n\times 1}{G^{(22)}} = -\mathbf{1}_{n\times 1}$
$\underset{n\times(d+1)}{G^{(2)}} = \begin{bmatrix} X & -\mathbf{1}_{n\times 1} \end{bmatrix}$
$\underset{n\times 1}{h^{(2)}} = \mathbf{y}$

(b.4) (0.25%) We want $G^{(3)}\mathbf{u} \preceq \mathbf{h}^{(3)} \iff \mathbf{y} - X\mathbf{w} \preceq \delta \cdot \mathbf{1_n}$. What should $G^{(3)} \in \mathbb{R}^{n\times(d+1)}$ and $h^{(3)} \in \mathbb{R}^n$ be?
Recall our third constraint is
$\mathbf{y} - X\mathbf{w} \preceq \delta \cdot \mathbf{1}_n \iff -X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq -\mathbf{y}$
We want

$$\underset{n\times(d+1)}{G^{(3)}} \cdot \mathbf{u} \preceq \underset{n\times 1}{\mathbf{h}^{(3)}}$$

We will now map our constraint into this form.

$$\begin{bmatrix} \underset{n\times d}{G^{(31)}} & \underset{n\times 1}{G^{(32)}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix} \preceq \underset{n\times 1}{\mathbf{h}^{(3)}} \iff -X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq -\mathbf{y}$$
$$\underset{n\times d}{G^{(31)}} \cdot \mathbf{w} + \underset{n\times 1}{G^{(32)}} \cdot \delta = -X\mathbf{w} - \delta \cdot \mathbf{1}_n \preceq -\mathbf{y} = \underset{n\times 1}{\mathbf{h}^{(3)}}$$

So,
$\underset{n\times d}{G^{(31)}} = -X$
$\underset{n\times 1}{G^{(32)}} = -\mathbf{1}_{n\times 1}$
$\underset{n\times(d+1)}{G^{(3)}} = \begin{bmatrix} -X & -\mathbf{1}_{n\times 1} \end{bmatrix}$
$\underset{n\times 1}{h^{(3)}} = -\mathbf{y}$

(b.5) (2%) Based on your derivations in (b), implement a Python function

$$\mathbf{w} = \text{minimizeLinf}(X, y)$$

that returns a $d \times 1$ vector of weights/parameters $\mathbf{w}$ corresponding to the solution of minimum L_∞ loss.

$$\mathbf{w} = \text{argmin}_{w\in\mathbb{R}^d} ||X\mathbf{w} - \mathbf{y}||$$

Based on our derivations in (b), we result in

$$\mathbf{c} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}$$

$$\mathbf{u} = \begin{bmatrix} \mathbf{w} \\ \delta \end{bmatrix} \in \mathbb{R}^{d+1}$$

$$G = \begin{bmatrix} \mathbf{0}_{1\times d} & -1 \\ X & -\mathbf{1}_n \\ -X & -\mathbf{1}_n \end{bmatrix} \in \mathbb{R}^{(2n+1)\times(d+1)}$$

$$\mathbf{h} = \begin{bmatrix} 0 \\ \mathbf{y} \\ -\mathbf{y} \end{bmatrix} \in \mathbb{R}^{2n+1}$$

$$\text{s.t.}$$

$$\underset{(2n+1)\times(d+1)}{G} \cdot \underset{(d+1)\times 1}{\mathbf{u}} \preceq \underset{(2n+1)\times 1}{\mathbf{h}}$$

Please see A1codes.py for full implementation.

(c) **(2%) Synthetic Regression Problem**
In this part, you will evaluate your implemented algorithms on a synthetic dataset.

(c.1) (1%) Implement a Python function
`train_loss, test_loss = synRegExperiments()`
that returns a 2 x 2 matrix train loss of average training losses and a 2 x 2 matrix test loss of average test losses (See Table 1 and Table 2 below.) It repeats 100 runs as follows
Please see **A1codes.py** for full details.

(c.2) (1%) Looking at your tables from above, analyze the results and discuss any findings you may have and the possible reason behind them.
Note that the matrices below are representative of the following table:

Model	Average L_2 loss	Average L_∞ loss
L_2 model L_∞ model		

Table 1: Table representation of loss data

After running `synRegExperiments` on the regression_train.csv and regression_test.csv data sets we get the following results:
For training data:

Model	Average L_2 loss	Average L_∞ loss
L_2 model	2.00305568	7.31711154
L_∞ model	2.36347873	6.6215357

Table 2: Training losses for L_2 and L_∞ models on regression_train.csv

For test data:

Model	Average L_2 loss	Average L_∞ loss
L_2 model	1.66071948	5.58874693
L_∞ model	1.9868478	5.02664782

Table 3: Test losses for L_2 and L_∞ models on regression_test.csv

For a randomly generated training and test data set using seed 101210291, we get the following results:

For training data:

Model	Average L_2 loss	Average L_∞ loss
L_2 model	0.11160734	1.68718399
L_∞ model	0.27832791	0.91119205

Table 4: Training losses for L_2 and L_∞ models on random data seeded with 101210291

For test data:

Model	Average L_2 loss	Average L_∞ loss
L_2 model	0.05252988	1.02181983
L_∞ model	0.35054548	2.26890905

Table 5: Test losses for L_2 and L_∞ models on random data seeded with 101210291

This shows that our L_2 loss, overall, has a lower average loss compared to the L_∞ loss. This suggests that the L_2 model fits the data better and will perform better compared to the L_∞ model for this population. It is shown with both the training and test data that the L_2 model has a lower average L_2 loss than the L_∞ loss. However, we see that in the L_2 model, the L_∞ loss is significantly worse than the L_∞ model, suggesting that there are a few individual points with high error.

(d) **(1.5%) Real-World Regression Problem**

(d.1) (1%) Here you will apply the linear regression algorithm to the concrete compressive strength (CCS) dataset. Please see **A1codes.py** for `preprocessCCS(dataset_folder)` implemenation.

(d.2) (0.5 %) Implement a Python function Please see **A1codes.py** for `runCCS(dataset_folder)` implemenation.
TODO: Add table of the results

Question 2: (7.5%) Logistic Regression
In this question, you will implement logistic regression, a classification method, and solve it using scipy.optimize.minimize. 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) **(2% Solving Convex Problem)**

(1.1) (1%) Before diving into logistic regression, let's first revisit the linear regression in Q1(a). Implement two python functions
Please see **A1codes.py** for the implemenation of `linearRegL2Obj(w, X, y)` and `linearRegL2Grad(w, X, y)`.

(2.2) (1%) Write a python function
Please see **A1codes.py** for the implemenation of `find_opt(obj_func, grad_func, X, y)`.

(b) **(2% Solving Logistic Regression)**
Implement two Python functions
Please see **A1codes.py** for the implemenation of `logisticRegObj(w, X, y)` and `logisticRegGrad(w X, y)`.

(c) **(2% Synthetic Binary classification Problem)**

(c.1) (1%) In this part, you will evaluate your implemenation on a synthetic dataset. Implement a Python function
Please see **A1codes.py** for the implemenation of `synClsExperiments()`.

After running `synClsExperiments` on the classification_train.csv and classification_test.csv data sets we get the following results:

· For training data we get an accuracy of 0.93

· For test data we get an accuracy of 0.925

For a randomly generated training and test data set using seed 101210291, we get the following results:

For training data:

`m` Train Accuracy	`dim1` Train Accuracy	`dim2` Train Accuracy
10 = 0.9795	1 = 0.8421	1 = 0.92535
50 = 0.923	2 = 0.93095	2 = 0.92585
100 = 0.92565	4 = 0.989	4 = 0.92755
200 = 0.925225	8 = 1.	8 = 0.93285

Table 6: Training accuracy for different hyper-parameters for random data seeded with 101210291

For test data:

`m` Test Accuracy	`dim1` Test Accuracy	`dim2` Test Accuracy
10 = 0.875145	1 = 0.83864	1 = 0.918075
50 = 0.9131	2 = 0.91621	2 = 0.916885
100 = 0.916855	4 = 0.969865	4 = 0.914395
200 = 0.919775	8 = 0.99356	8 = 0.90897

Table 7: Test accuracy for different hyper-parameters for random data seeded with 101210291

(c.2) (1%) Looking at your tables from above, analyze the results and discuss any findings you may have and the possible reasons behind them

The initial testing done on the real world data from the csv files shows that our logistic regression model is performing well with minimal accuracy drop when seeing new (test) data.

In our randomized dataset (seeded with 101210291), we see that as we introduce more training data (`m`), the training and test accuracies both improve. Initially, the training accuracy is high ($\approx 97\%$) for `m` $= 10$, but is lower for training accuracy ($\approx 87\%$). This suggests that the model is overfitting for a small amount of training data. The results show that the as we train on more data, the model becomes more stable, and test accuracy improves. Furthermore, as we increase the number of features, specified in `dim1`, we see that both training and test accuracies trend towards 1. This shows that the more classification data we can train on, the better our model will be at predicting. Lastly, we see that in `dim2`, the accuracies stay more steady as the number of features increase. That is becuase these features are not weighted (-1/+1) like we do in `dim1` demonstrating that features that don't provide valuable information don't increase our accuracy.

(d) **(1.5%) Real-World Binary Classification Problem**

(d.1) (1%) Now you will apply logistic regression a real-world problem, the Breast Cancer Wisconsin (BCW) dataset

To start, you need to preprocess the data. Implement a Python function

Please see **A1codes.py** for `preprocessBCW(dataset_folder)` implemenation.

(d.2) (0.5%) Implement a Python function
Please see **A1codes.py** for `runBCW(dataset_folder)` implemenation.