Table of Contents

Question 1

(a)

Question 1

a) to do all of question 1 it is worth mentoring

$$p(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

counts A B C D

$+ = 5, 3, 9, 6 \rightarrow$ total $= 23$

$- = 11, 3, 0, 2 \rightarrow$ total $= 16$

we now have our parameters

$\theta_+ = (\frac{5}{23}, \frac{3}{23}, \frac{9}{23}, \frac{6}{23})$

$\theta_- = (\frac{11}{16}, \frac{3}{16}, \frac{0}{16}, \frac{2}{16})$

using this we can compute $P(x_*|+)$, $x_* = (1,0,0,2)$

$$p(x_*|+) = 3! \cdot \frac{(5/23)^1}{1!} \cdot \frac{(3/23)^0}{0!} \cdot \frac{(5/23)^0}{0!} \cdot \frac{(6/23)^2}{2!}$$

$$= \frac{540}{12167}$$

$$p(+|x_*) = \frac{P(x_*|+) \cdot P(+)}{P(x_*)}$$

total words in document = 39

total A = 16

total B = 6

total C = 9

total D = 8

$p(+) = 4/8 \rightarrow |D| = 8 \quad |D_+| = 4$

all we miss is $p(x_*)$

$$p(x_*) = \frac{\left(\sum_{i=1}^{4} x_i\right)!}{\prod_{i=1}^{4} x_i!} \cdot \prod_{i=1}^{4} p_i^{x_i} = \frac{3!}{1! \cdot 2!} \cdot \left(\frac{16}{39}\right)^1 \cdot \left(\frac{6}{39}\right)^0 \cdot \left(\frac{9}{39}\right)^0 \cdot \left(\frac{8}{39}\right)^2$$

$$= \frac{1024}{19773}$$

$$\boxed{\text{So } p(+|x_*) = \frac{\left(\frac{540}{12167}\right) \cdot \frac{1}{2}}{\left(\frac{1024}{19773}\right)} = \frac{2669355}{6229504} = 0.4285}$$

(b)

Question 1

b) if we apply add-1 Smoothing we get the following new table

document no.	A	B	C	D	class
1	2	0	4	4	+
2	0	3	3	0	+
3	3	0	0	2	+
4	0	0	2	0	+
P_1	1	0	0	0	+
P_2	0	1	0	0	+
P_3	0	0	1	0	+
P_4	0	0	0	1	+
5	0	0	0	1	−
6	3	0	0	0	−
7	4	3	0	0	−
8	4	0	0	1	−
P_5	1	0	0	0	−
P_6	0	1	0	0	−
P_7	0	0	1	0	−
P_8	0	0	0	1	−

now we have new counts as

A B C D

$+ = 6, 4, 10, 7 \rightarrow$ total = 27

$- = 12, 4, 1, 3 \rightarrow$ total 20

total words = 47

total A = 18

total B = 8

total C = 11

total D = 10

we now have our parameters as

$$\theta_+ = \left(\frac{6}{27}, \frac{4}{27}, \frac{10}{27}, \frac{7}{27}\right)$$

$$\theta_- = \left(\frac{12}{20}, \frac{4}{20}, \frac{1}{20}, \frac{3}{20}\right)$$

using this we can compute $p(x_+|-)$

$$p(x_+|-) = 3! \cdot \frac{\left(\frac{12}{20}\right)^1}{1!} \cdot \frac{\left(\frac{4}{20}\right)^0}{0!} \cdot \frac{\left(\frac{1}{20}\right)^0}{0!} \cdot \frac{\left(\frac{3}{20}\right)^2}{2!}$$

$$= \frac{81}{2000}$$

Question 1

B) $p(-) = \frac{1}{2}$ $|D| = 16, |D_-| = 8$

$p(x_+)$ ~~[illegible]~~ need to recompute with smoothing

$$p(x_+) = \frac{3!}{1! \cdot 2!} \cdot \left(\frac{18}{47}\right)^1 \cdot \left(\frac{8}{47}\right)^0 \cdot \left(\frac{11}{47}\right)^0 \cdot \left(\frac{10}{47}\right)^2$$

$$= \frac{5400}{103823}$$

now we get

$$\boxed{\begin{aligned} p(-|x_+) &= \frac{\left(\frac{81}{2000}\right) \cdot \frac{1}{2}}{\left(\frac{5400}{103823}\right)} \\ &= \frac{311469}{800000} \\ &= 0{,}3893 \end{aligned}}$$

(c)

Question 1

(c) to work with bernoulli NB we would prefer a new table

document no.	A	B	C	D	class
1	1	0	1	1	+
2	0	1	1	0	+
3	1	0	0	1	+
4	0	0	1	0	+
5	0	0	0	1	−
6	1	0	0	0	−
7	1	1	0	0	−
8	1	0	0	1	−

now we get the following counts

$$\begin{array}{lcl} & A\ B\ C\ D & \\ + = & 2,1,3,2 & \longrightarrow +\text{documents} = 4 \\ - = & 3,1,0,2 & \longrightarrow -\text{documents} = 4 \end{array}$$

total$_d$ = 8
total A = 5
total B = 2
total C = 3
total D = 4

we can compute

$\Theta_+ = (2/4, 1/4, 3/4, 2/4)$

$\Theta_- = (3/4, 1/4, 0/4, 2/4)$

$x_* = (1,0,0,1)$

using this we can compute $P(x_* | +)$

$$p(x_*|+) = (2/4)\cdot(1-1/4)\cdot(1-3/4)\cdot(2/4) = 3/64$$

$p(+) = 1/2$ since $|D| = 8$, $|D_+| = 4$

$$p(x_*) = \prod_{i=0}^{4} p^{\mathbb{I}(x_i=1)}(1-p)^{\mathbb{I}(x_i=0)}$$ where $\mathbb{I}$ is indicator function for a boolean

$$p(x_*) = \left(\frac{5}{8}\right)\cdot\left(1-\frac{2}{8}\right)\cdot\left(1-\frac{3}{8}\right)\cdot\left(4/8\right) = 75/512$$

now we get $p(+|x_*) = \dfrac{3/64 \cdot \frac{1}{2}}{\left(\frac{75}{512}\right)} = 4/25$

$\boxed{= 0.16}$

(d)

(d) applying smoothing gives the following table

Question 1

document no.	A	B	C	D
1	1	0	1	1
2	0	1	1	0
3	1	0	0	1
4	0	0	1	0
P_1	1	1	1	1
P_2	0	0	0	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0
8	1	0	0	~~0~~ 1
P_3	1	1	1	1
P_4	0	0	0	0

$\text{total}_d = 22$
total A = 7
total B = 4
total C = 5
total D = 6

now we get the following ~~results~~ counts

$+ = \overset{A}{3}, \overset{B}{2}, \overset{C}{4}, \overset{D}{3} \rightarrow$ + documents = 6

$- = 4, 2, 1, 3$ - documents = 6

we can compute

$$\theta_+ = (3/6, 2/6, 4/6, 3/6) \qquad x^* = (1, 0, 0, 1)$$

$$\theta_- = (4/6, 2/6, 1/6, 3/6)$$

using this we can compute

$$p(x_*|-) = (4/6)\cdot(1-2/6)\cdot(1-1/6)\cdot(3/6) = 5/27$$

$p(-) = 1/2 \qquad |D| = 12 \qquad |D_-| = 6$

$$p(x_*) = (7/22)\cdot(1-4/22)\cdot(1-5/22)\cdot(6/22) = 49/432$$

$$p(-|x_*) = \frac{5/27 \cdot \frac{1}{2}}{(49/432)}$$

$$\boxed{= 0.8163}$$

Question 2

(a)

With respect to w0 the partial derivative was as follows

(a) we have

Question 2

$$\mathcal{L}_c(y,\hat{y}) = \sum_{i=1}^{n} \left[\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{\frac{1}{2}} - 1 \right]$$

$$\frac{\partial \mathcal{L}_c(y_i,\hat{y}_i)}{\partial w_0} = \frac{\partial}{\partial w_0}\sum_{i=1}^{n} \left[\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{\frac{1}{2}} - 1 \right]$$

$$= \sum_{i=1}^{n} \left[\frac{\partial}{\partial w_0}\left(\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{\frac{1}{2}} - 1\right) \right] \quad ①$$

② $$= \sum_{i=1}^{n} \left[\frac{1}{2}\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{-\frac{1}{2}} \cdot (-2)\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)\right) \right]$$

$$= \sum_{i=1}^{n} \left[-\frac{y_i - w_0 - w_1 x_i}{c^2\sqrt{\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1}} \right]$$

With respect to w1

to go from ① -> ② we take derivative of whole sqrt and multiply by derivative of inside

chain rule

we had $\sqrt{(f)^2} \rightarrow \frac{1}{2} \cdot \frac{1}{\sqrt{(f)^2}} \cdot$ derivative of $(f)^2$

Similarly $$\frac{\partial \mathcal{L}_c(y_i,\hat{y}_i)}{\partial w_1} = \frac{\partial}{\partial w_1}\sum_{i=1}^{n} \left[\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{\frac{1}{2}} - 1 \right]$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial w_1}\left[\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{\frac{1}{2}} - 1 \right]$$

$$= \sum_{i=1}^{n} \left[\frac{1}{2}\left(\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1\right)^{-\frac{1}{2}} \cdot \frac{2}{c^2}(-x_i)(y_i - w_0 - w_1 x_i) \right]$$

$$= \sum_{i=1}^{n} \left[\frac{-x_i(y_i - w_0 - w_1 x_i)}{c^2\sqrt{\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1}} \right]$$

$$= \sum_{i=1}^{n} \left[\frac{-x_i y_i + w_0 x_i + w_1 x_i^2}{c^2\sqrt{\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1}} \right]$$

I took it a bit further and pushed the sum into the result for simplification and the result is as follows

Question 2

a) (continued)

to simplify even further and remove sum we will create new variable

let $\mathring{x} = \sum_{i=1}^{n} x_i$

and $p = \frac{1}{c^2\sqrt{\frac{1}{c^2}(\mathring{y} - nw_0 - w_1\mathring{x})^2 + 1}}$

So now $\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_0} = \sum_{i=1}^{n} \left[\frac{-y_i - w_0 - w_1 x_i}{c^2\sqrt{\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1}} \right]$

by shifting in sigma and splitting the fraction and substituting p, we get

$$\boxed{\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_0} = \frac{-\mathring{y}}{p} - \frac{nw_0}{p} - \frac{w_1\mathring{x}}{p}}$$

similarly for $\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_1} = \sum_{i=1}^{n} \left[\frac{-x_i y_i + w_0 x_i + w_1 x_i^2}{c^2\sqrt{\frac{1}{c^2}(y_i - w_0 - w_1 x_i)^2 + 1}} \right]$

by shifting in sigma and splitting the fraction and substituting p, we get

$$\boxed{\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_1} = \frac{-\mathring{xy}}{p} + \frac{w_0\mathring{x}}{p} + \frac{w_1\mathring{x^2}}{p}}$$

So final answer

$$\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_0} = \frac{-\mathring{y}}{p} - \frac{nw_0}{p} - \frac{w_1\mathring{x}}{p}$$

$$\frac{\partial \mathcal{L}_c(y,\hat{y})}{\partial w_0} = \frac{-\mathring{xy}}{p} + \frac{w_0\mathring{x}}{p} + \frac{w_1\mathring{x^2}}{p}$$

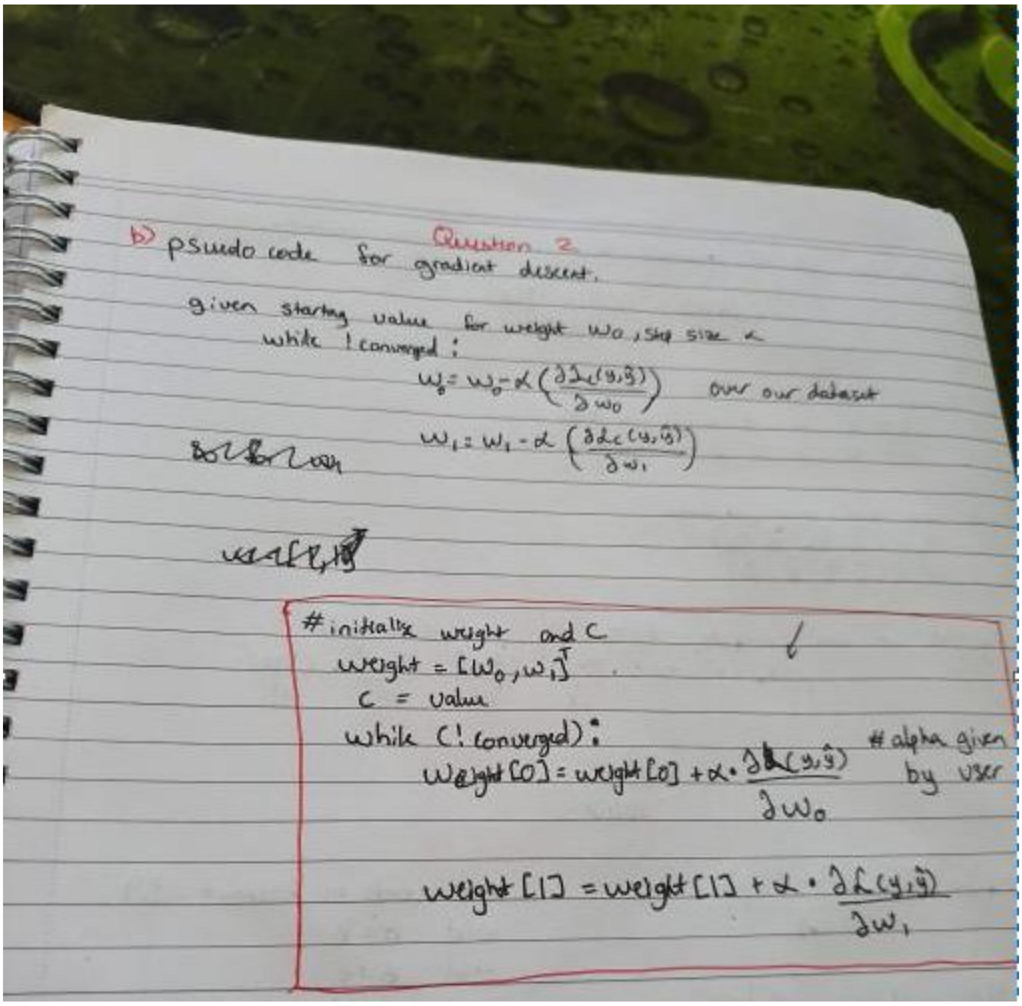
where a variable $\mathring{i} = \sum_{j=1}^{n} i_j$ and $p = \frac{1}{c^2\sqrt{\frac{1}{c^2}(\mathring{y} - w_0 n - w_1\mathring{x})^2 + 1}}$ //

Note this is just taking the answer a bit further for readability.

(b)

(b) [**2 marks**] Using the gradients computed in (a), write down the gradient descent updates for w_0 and w_1 (using pseudocode), assuming a step size of α. Note that in gradient descent we consider the loss over the entire dataset, not just at a single observation. For simplicity, assume that your updates are always based on the values at the previous time step, even if you have access to the value at the current time step (i.e., when updating multiple parameters, the update for $w_1^{(t+1)}$ might depend on w_0, so you could use the new value of w_0, $w_0^{(t+1)}$, since it has already been computed, but here we assume that we just use the old value $w_0^{(t)}$).

Under the assumption we run the algorithm until convergence, I would do something along the lines of:



Question 2

b) psudo code for gradiant descent.

given starting value for weight w_0, step size α

while !converged:

$$w_0 = w_0 - \alpha\left(\frac{\partial L(y,\hat{y})}{\partial w_0}\right) \quad \text{over our dataset}$$

$$w_1 = w_1 - \alpha\left(\frac{\partial L(y,\hat{y})}{\partial w_1}\right)$$

```
#initialise weight and C
weight = [w0, w1]^T
C = value
while (!converged):
    weight[0] = weight[0] + α · ∂L(y,ŷ)/∂w0     # alpha given by user
    weight[1] = weight[1] + α · ∂L(y,ŷ)/∂w1
```

(c)

The code used to solve this problem also found in solutions.py is the following

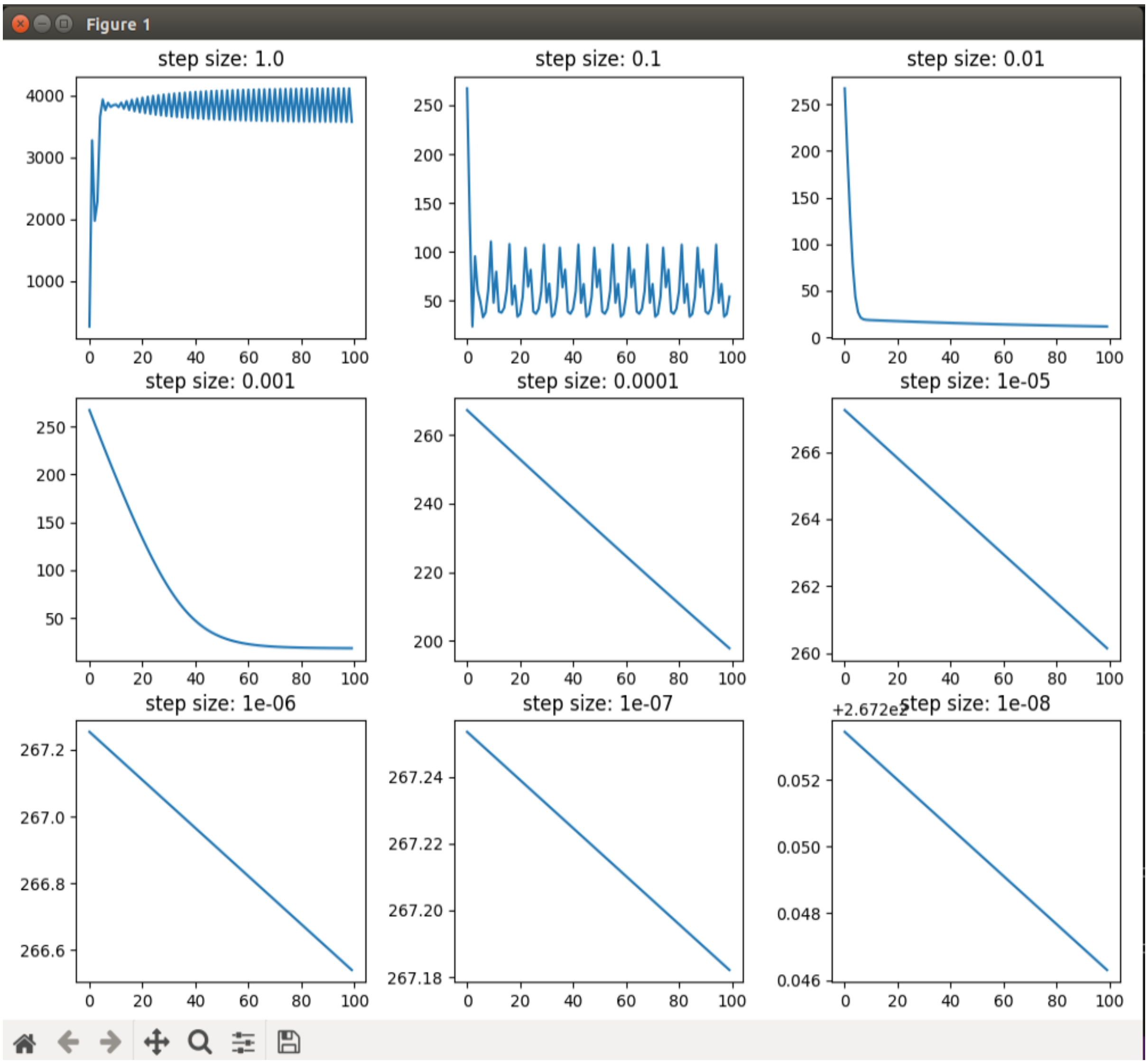
```python
## (c)

# YOUR CODE HERE
losses_2c = np.zeros((9,100))
# moving provided code
alphas = [10e-1, 10e-2, 10e-3,10e-4,10e-5,10e-6,10e-7, 10e-8, 10e-9]

def question_2c(alphas):
    c_value_2c = 2
    for i in range(9):
        weights_2c = [1, 1]
        for j in range(100):
            loss = loss_total(c_value_2c, weights_2c, x, y)
            weights_2c[0] = weights_2c[0] - alphas[i]*descent_w0(c_value_2c, weights_2c,x, y)
            weights_2c[1] = weights_2c[1] - alphas[i]*descent_w1(c_value_2c, weights_2c,x, y)
            losses_2c[i][j] = loss

# assuming |x| = |y|
def loss_total(c, weights, x, y):
    sum_value = 0
    for i in range(len(x)):
        c_squared = (1)/(c**2)
        inside_bracket = (y[i] - weights[0] - weights[1]*x[i])
        sum_value += np.sqrt(c_squared*(inside_bracket**2)+ 1) - 1
    return sum_value

def descent_w0(c, w, x, y):
    sum_value = 0
    for i in range(len(x)):
        c_squared = (1)/(c**2)
        inside_bracket = (y[i] - w[0] - w[1]*x[i])
        sum_value += (-inside_bracket)/((c**2)*np.sqrt(c_squared*(inside_bracket**2)+1))
    return sum_value

def descent_w1(c, w, x, y):
    sum_value = 0
    for i in range(len(x)):
        c_squared = (1)/(c**2)
        inside_bracket = (y[i] - w[0] - w[1]*x[i])
        sum_value +=(-x[i]*(inside_bracket)/((c**2)*np.sqrt((c_squared*(inside_bracket**2) + 1))))
    return sum_value


# uncomment for 2c, comment for 2e
# plotting help
fig, ax = plt.subplots(3,3, figsize=(10,10))
question_2c(alphas)
for i, ax in enumerate(ax.flat):
    # losses is a list of 9 elements. Each element is an array of length 100 storing the loss at each iteration for that particular step size
    ax.plot(losses_2c[i])
    ax.set_title(f"step size: {alphas[i]}")  # plot titles
plt.tight_layout()      # plot formatting
plt.savefig("grid_plot.png")
plt.show()
```

It is worth noting I moved the initalisation of alphas at the start from the plotting, to make the flow of code nicer, and all the components were functionalised for generality and potential use in other places. Also, I put a lot of parameters to make the code more readable and easier to work with, it is very hard to work with messy code.

(d)

[1 mark] Comment on your results in part (c): what do you see happening for different step sizes? Why does this occur?

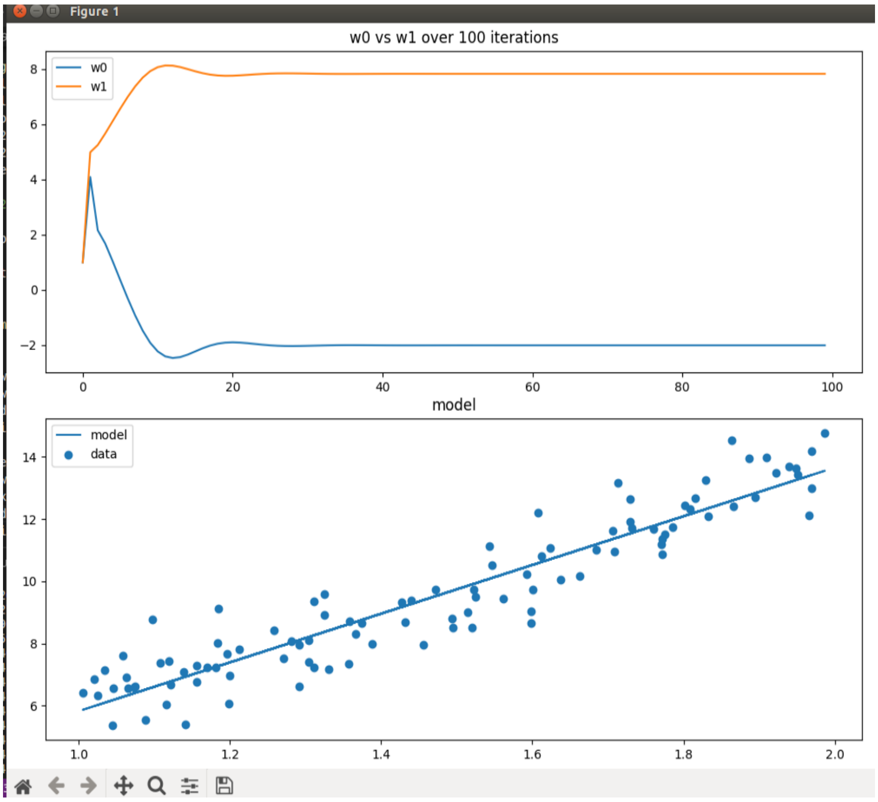
The following was output by running the code



What we can see is the following, for step sizes between 0.001 and 0.1, our model's loss would converge to specific values, (depending on the value of our step of course), for models with a lower value than 0.001, their loss would stay almost the same throughout each iteration, this could be due to making almost negligible changes to our weights and since we are only doing 100 iterations we don't converge like the other models. For models with a value higher than 0.1 we observed the error would increase drastically and converge at a higher error. This is due to making a more complex value, with higher weights since the increase/decrease is much larger. It seems that for the case of 100 iterations a model with step size between 0.001 and 0.1 would be optimal, as it would converge to a lower loss.

(e)

(e) [3 marks] To find an appropriate step-size, re-run the gradient descent to find the optimal model parameters. State this step-size, the final model, and generate two plots: the first for the weights w0 and w1 over the 100 iterations, and the second of the data with the final model super-imposed. What do you think of the final model? Are there any obvious ways to improve it? Be sure to include the plots in your answers PDF file, and the code used in your solutions.py file.



In the optimal model after multiple run throughs tweaking values I had

- Alpha = 0.065
- Final model y = -2.00547588 + (7.82950538) * x

The final model I created seems to be a linear fit as best as possible for the data shown (almost like it goes through the centre of the data cluster). The problem with the model and why it cannot do really well on this data, is because it seems that the inductive bias of linear regression requires the true model to be representable in a linear function, but from looking at the data, it seems to be non-linear, potentially polynomial.

One way that we can improve the model would be to play around with starting parameters, potentially even c, another way would be to remove potential noise/outliers from the data aswell.

Code shown here

```
weight_plot = np.zeros((2,100))
losses_2e = np.zeros((1,100))
def quesiton_2e():
    c_value_2e = 2
    weights_2e = [1, 1]
    for i in range(100):
        weight_plot[0][i] = weights_2e[0]
        weight_plot[1][i] = weights_2e[1]
        loss = loss_total(c_value_2e, weights_2e, x, y)
        weights_2e[0] = weights_2e[0] - (0.065)*descent_w0(c_value_2e, weights_2e, x, y)
        weights_2e[1] = weights_2e[1] - (0.065)*descent_w1(c_value_2e, weights_2e, x, y)
        losses_2e[0][i] = loss

# uncomment for 2c, comment for 2e
# plotting help
fig, ax = plt.subplots(2,1, figsize=(10,10))
quesiton_2e()
print(weight_plot)
print("______")
print(losses_2e)
for i, ax in enumerate(ax.flat):
    if(i == 0):
        # losses is a list of 9 elements. Each element is an array of length 100 storing the loss at each iteration for that particular step size
        ax.plot(weight_plot[0], label = 'w0')
        ax.plot(weight_plot[1], label = 'w1')
        ax.legend()
        ax.set_title(f"w0 vs w1 over 100 iterations")    # plot titles
    else:
        ax.scatter(x,y, label = 'data')
        model = weight_plot[0][99] + weight_plot[1][99]*x
        ax.plot(x, model, label = 'model')
        ax.legend()
        ax.set_title(f"model")   # plot titles


plt.tight_layout()      # plot formatting
plt.savefig("grid_plot.png")
plt.show()
```

(f)

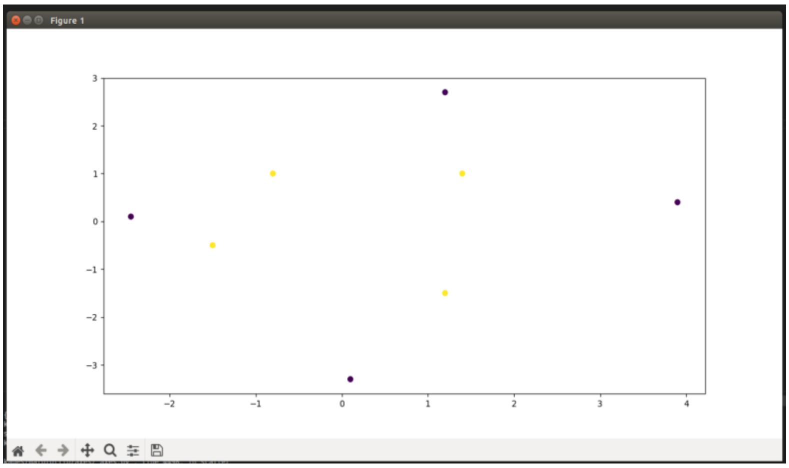
Consider the following scenario: you re-run the analysis for various values of c, and you notice that the optimal step-size varies across different values of c. Describe an approach to choosing an appropriate value of c.

A suitable approach to choosing an appropriate value of c would be using K-cross-fold validation where we partition the data into training set and validation set, using a different subset each time, using this we are able to run multiple tests on a model and see the performance for each model. We are also able to see trends in the models this way and pick a value for c that has the lowest loss across all models.

Question 3

(a)

Plotting the original data using matplotlib we get the following



As we can see, this is not linearly separable, so we need to increase the dimension of our feature space

I was unable to do part (a) or (b) but I think using cross-validation and checking would have been the best way to find the minimum, I just did not know how to implement in NumPy.

(c)

I could not get the answer to (a) and (b) so I tried to make the perceptron algorithm using the initial feature space (however it does not converge) and attempted to make it as general as possible to apply to a case where I could get (a) and (b). the following code was used (also to print table). Note if I were able to get (a) I would use the linearly separable space and simply apply the p weights to the p features in the new space. This is a more general approach of how I would of handled it had I known how to achieve (a).

```python
## Question 3

# (c)
# YOUR CODE HERE
def train_perceptron():
    weights_3c = [1, 1]
    learning_rate_3c = 0.2
    epochs = 0
    converged = False
    x1s_3c = [-0.8, 3.9, 1.4, 0.1, 1.2, -2.45, -1.5, 1.2]
    x2s_3c = [1, 0.4, 1, -3.3, 2.7, 0.1, -0.5, -1.5]
    ys_3c = [1, -1, 1, -1, -1, -1, 1,1]

    print("Iteration No. | ", " w0                          |", " w1                          | ")
    print("---------------------------------------------------------------------------------------")
    change_in_weight_0 = 0
    change_in_weight_1 = 0
    while(converged == False and epochs < 100):
        print(epochs, " | ", weights_3c[0], " + ",change_in_weight_0, "                          | " ,weights_3c[1], " + ",change_in_weight_1)
        change_in_weight_0 = 0
        change_in_weight_1 = 0
        converged = True
        for i in range(len(x1s_3c)):
            if ((weights_3c[0]*x1s_3c[i] + weights_3c[1]*x2s_3c[i]) * ys_3c[i]) < 0:

                change_in_weight_0 += learning_rate_3c*ys_3c[i]*(x1s_3c[i])
                change_in_weight_1 += learning_rate_3c*ys_3c[i]*(x2s_3c[i])

                weights_3c[0] = weights_3c[0] + learning_rate_3c*ys_3c[i]*(x1s_3c[i])
                weights_3c[1] = weights_3c[1] + learning_rate_3c*ys_3c[i]*(x2s_3c[i])
                converged = False
        epochs += 1

    print("---------------------------------------------------------------------------------------")
train_perceptron()
```

Outputs

```
65 |  -0.01000000000000345  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
66 |  0.0599999999999965  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
67 |  0.10999999999999643  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
68 |  -0.010000000000003617  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
69 |  0.059999999999996334  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
70 |  0.10999999999999627  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
71 |  -0.010000000000003784  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
72 |  0.05999999999999617  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
73 |  0.1099999999999961  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
74 |  -0.01000000000000395  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
75 |  0.059999999999996  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
76 |  0.10999999999999593  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
77 |  -0.010000000000004117  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
78 |  0.059999999999995834  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
79 |  0.10999999999999577  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
80 |  -0.010000000000004283  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
81 |  0.05999999999999567  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
82 |  0.1099999999999956  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
83 |  -0.01000000000000445  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
84 |  0.0599999999999955  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
85 |  0.10999999999999543  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
86 |  -0.010000000000004616  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
87 |  0.059999999999995335  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
88 |  0.10999999999999527  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
89 |  -0.010000000000004783  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
90 |  0.05999999999999517  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
91 |  0.1099999999999951  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
92 |  -0.01000000000000495  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
93 |  0.059999999999995  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
94 |  0.10999999999999494  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
95 |  -0.010000000000005116  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
96 |  0.059999999999994835  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
97 |  0.10999999999999477  +  0.04999999999999993                     |  -0.24000000000000024  +  0.4
98 |  -0.010000000000005282  +  -0.12000000000000005                     |  -0.3800000000000002  +  -0.14
99 |  0.05999999999999467  +  0.06999999999999995                     |  -0.6400000000000002  +  -0.26
---------------------------------------------------------------------------------------
bob@Bob ~>
```

Can clearly see for my case it was not converging because non-linearly separable feature space.

(d)

Question 3

d) $k(x,y) = (2x^T y + 3)^3$

$= 8x^{3T}y^3 + 3(2x^T y)^2 \cdot 3 + 3(2x^T y)(3)^2 + 3^3$

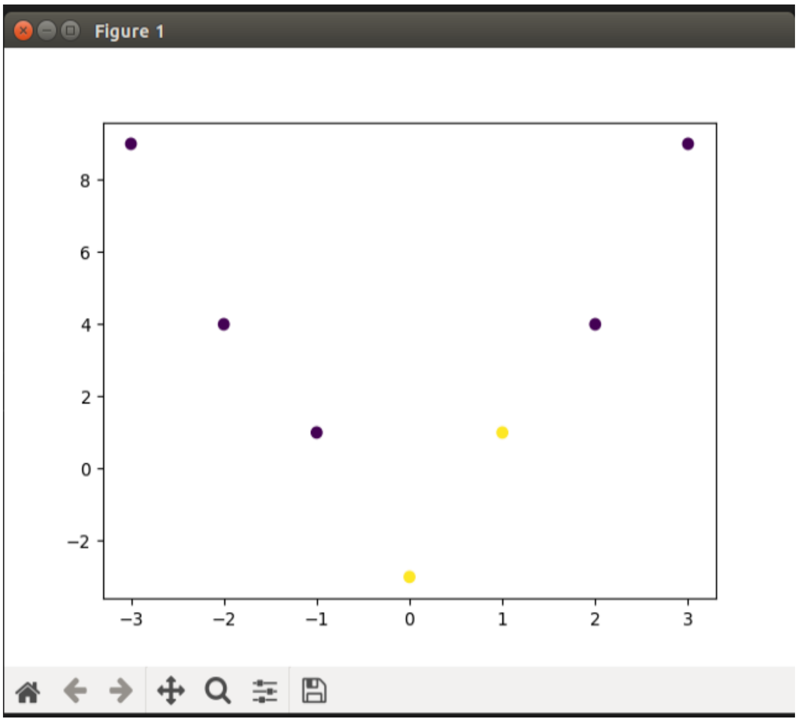
$\boxed{= 8x^{3T}y^3 + 36x^{T2}y^2 + 54x^T y + 9}$ //

(I skipped e could not do it)

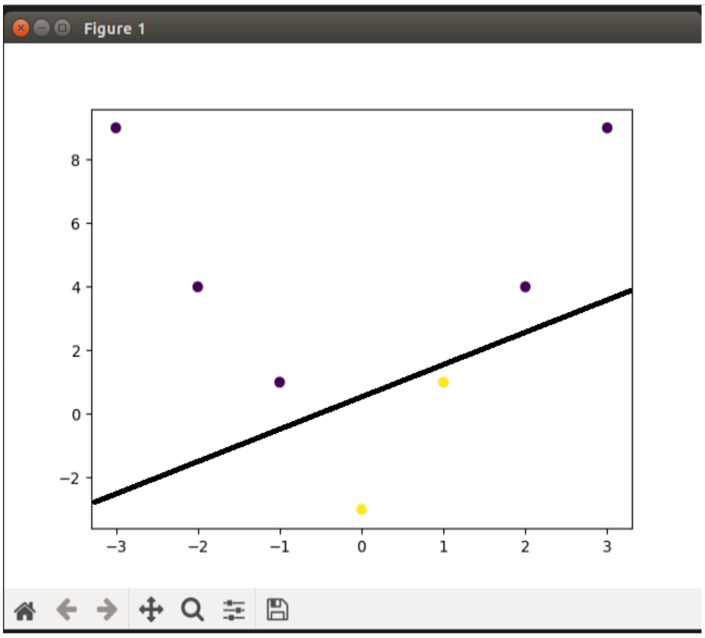
Question 4

(a)

Plot of data using matplotlib



Yes, the data is linearly separable, we can construct a line y = mx + b such that it separates all yellow circles from all purple circles as follows in figure below



(b)

b) Question 4)

GRAM MATRIX

$$X' = \begin{pmatrix} 3 & -9 \\ 2 & -4 \\ 1 & -1 \\ 0 & -3 \\ 1 & 1 \\ -2 & -4 \\ -3 & -9 \end{pmatrix} \qquad X'^T = \begin{pmatrix} 3 & 2 & 1 & 0 & 1 & -2 & -3 \\ -9 & -4 & -1 & -3 & 1 & -4 & -9 \end{pmatrix}$$

$$G = X'X'^T = \begin{bmatrix} 90 & 42 & 12 & 27 & -6 & 30 & 72 \\ 42 & 20 & 6 & 12 & -2 & 12 & 30 \\ 12 & 6 & 2 & 3 & 0 & 2 & 6 \\ 27 & 12 & 3 & 9 & -3 & 12 & 27 \\ -6 & -2 & 0 & -3 & 2 & -6 & -12 \\ 30 & 12 & 2 & 12 & -6 & 20 & 42 \\ 72 & 30 & 6 & 27 & -12 & 42 & 90 \end{bmatrix}$$

we can also see from the plot which points are the support vectors
moment of realisation i found a 7x7 gram matrix for no reason
we can ignore all points which aren't the support vectors
and our margin won't change. (from Kernel lecture)

now we get

x_1	x_2	y
1	1	1
2	4	-1

← seen in plot as support vectors

now $X' = \begin{pmatrix} 1 & 1 \\ -2 & -4 \end{pmatrix}$ $\quad X'^T = \begin{pmatrix} 1 & -2 \\ 1 & -4 \end{pmatrix}$

$$G = X'X'^T = \begin{bmatrix} 2 & -6 \\ -6 & 20 \end{bmatrix}$$

Continued Question 4

by using our Gram Matrix G

$$G = \begin{bmatrix} 2 & -6 \\ -6 & 20 \end{bmatrix}$$

we just need to find

$$\arg\max_{\alpha_1,\alpha_2} \sum_{i=1}^{2} \alpha_i - \frac{1}{2}\sum_{i=1}^{2}\sum_{i=1}^{2} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

under the condition $\alpha_i \geq 0$ for all i, $\sum_{i=1}^{2} \alpha_i y_i = 0$

one of our constraints can be written as $\alpha_1 - \alpha_2 = 0$

$\alpha_1 = \alpha_2$

now we get

$$\arg\max_{\alpha_1\alpha_2} \left(2\alpha_1 - \frac{1}{2}\left(2\alpha_1^2 - 6\alpha_1\alpha_2 - 6\alpha_2\alpha_1 + 20\alpha_2^2\right)\right)$$

we know $\alpha_1 = \alpha_2$ from our constraint

$$\arg\max_{\alpha_1,\alpha_2} \left(2\alpha_1 - \frac{1}{2}\left(2\alpha_1^2 - 6\alpha_1^2 - 6\alpha_1^2 + 20\alpha_1^2\right)\right)$$

$$\arg\max_{\alpha_1} \left(2\alpha_1 - 5\alpha_1^2\right)$$

by taking derivative

$$\frac{\partial}{\partial \alpha_1}\left(2\alpha_1 - 5\alpha_1^2\right) = 2 - 10\alpha_1$$

Solve when $=0$

$0 = 2 - 10\alpha_1$

$\boxed{\alpha_1 = 1/5}$

since $\alpha_1 = \alpha_2$ → $\boxed{\alpha_2 = 1/5}$

(c)

Question 4

c) weight vector

$$w = \alpha_1 x_1 - \alpha_2 x_2$$

$$= \tfrac{1}{5}(x_1 - x_2)$$

$$= \tfrac{1}{5}\left(\begin{pmatrix}1\\1\end{pmatrix} - \begin{pmatrix}2\\4\end{pmatrix}\right)$$

$$= \tfrac{1}{5}\begin{pmatrix}-1\\-3\end{pmatrix}$$

$$w = \begin{pmatrix}-1/5\\-3/5\end{pmatrix}$$

$$\|w\| = \sqrt{(-\tfrac{1}{5})^2 + (-\tfrac{3}{5})^2}$$

$$\|w\| = \sqrt{\tfrac{1}{25} + \tfrac{9}{25}}$$

$$= \frac{\sqrt{10}}{5}$$

Using this we know margin $= \dfrac{1}{\|w\|} = \dfrac{\sqrt{10}}{2}$

we also know $y_i \cdot (w \cdot x_i - t) = 1$ $(m = 1)$

Using instance 1 $1 \cdot (\underbrace{\cancel{-4/5}}_{}\overset{w \cdot x_1}{} - t) = 1$

$$1 = 1 \cdot (-\tfrac{4}{5} - t) \rightarrow t = -\tfrac{9}{5}$$

$$\boxed{w = \begin{pmatrix}-1/5\\-3/5\end{pmatrix} \quad \text{margin} = \frac{\sqrt{10}}{2} \quad \text{threshold} = -\tfrac{9}{5}}$$

(d)

(d) [2 marks] Discuss briefly the following claim: Linear classifiers are unable to represent non-linear functions.

The following claim is not true, you are able to use linear classifiers to represent non-linear functions by increasing the feature space in a higher dimensionality, whilst still being linear in that dimensionality. This is commonly seen in Kernel functions which help map non-linear inputs into a feature space that can linearly separate with a hyperplane.

(e)

(e) [3 marks] In your own words, explain what is meant by the 'kernel trick'. Why is this such an important technique for machine learning?

The kernel trick is an optimisation that extends beyond just support vector machines. In the most basic sense, It allows us to obtain the results of performing our operations on our new feature space, without having to instantiate the new feature space. It allows us to compute dot products in new feature spaces from our original feature space without creating new feature spaces. This is extremely important in terms of efficiency, since we don't have to recompute, we are able to make much faster calculations. This technique has been adopted by other aspects of machine learning not just SVM's as it lets us transform data into higher dimensions (feature spaces) in an efficient way.

Question 5

(a)

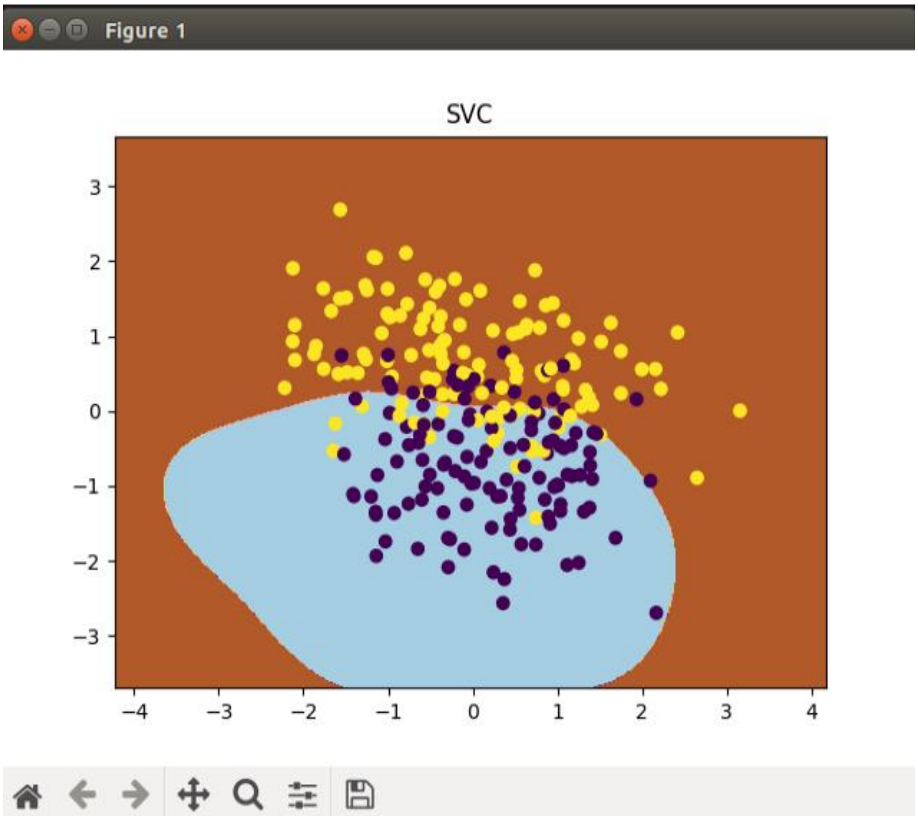


Figure 1
SVC
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

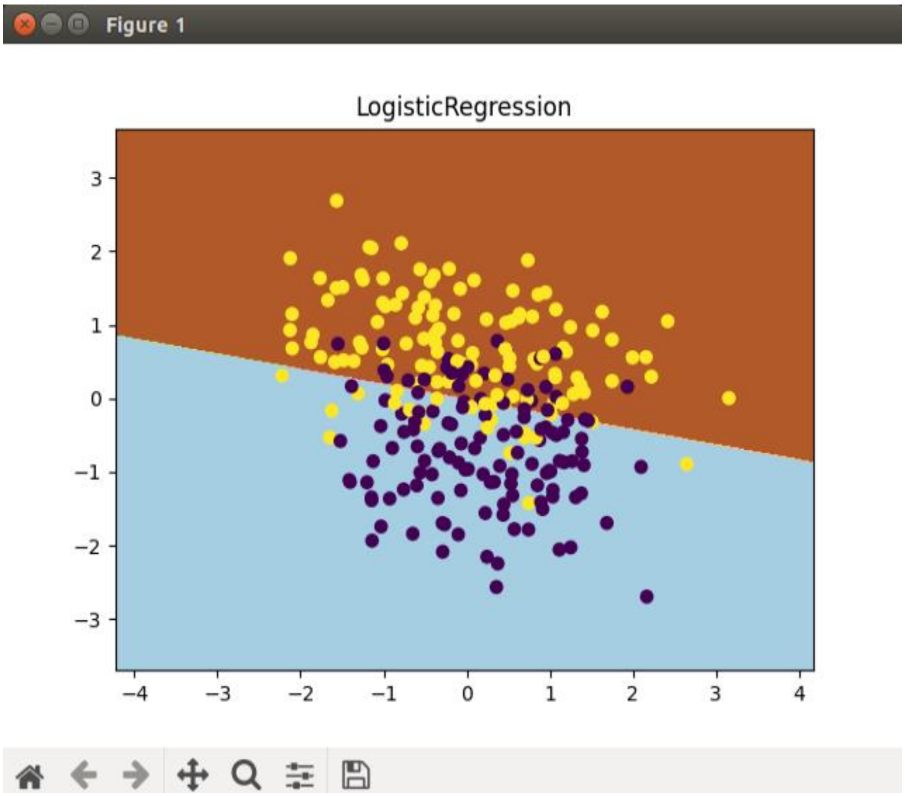


Figure 1
LogisticRegression
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

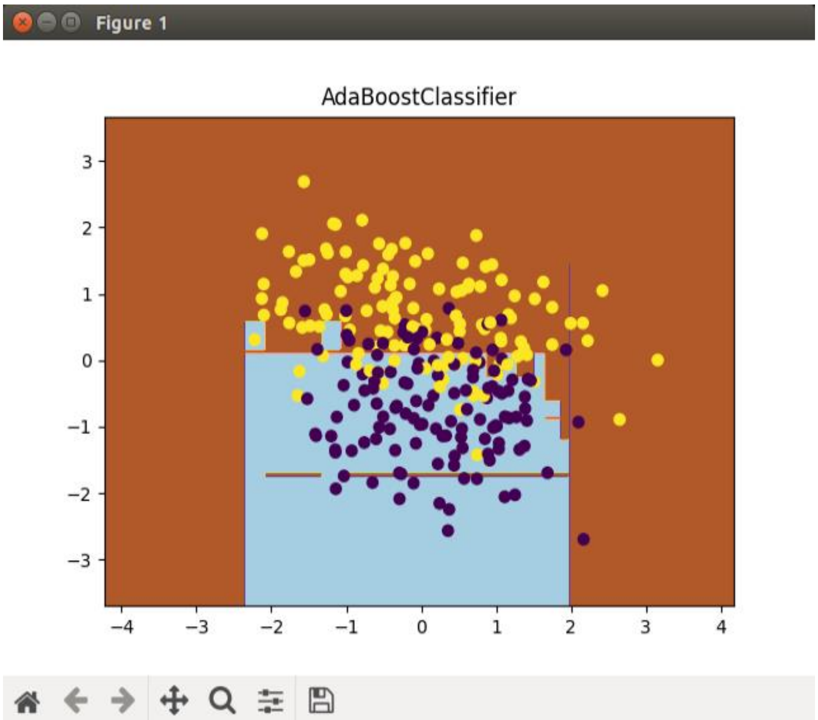


Figure 1
AdaBoostClassifier

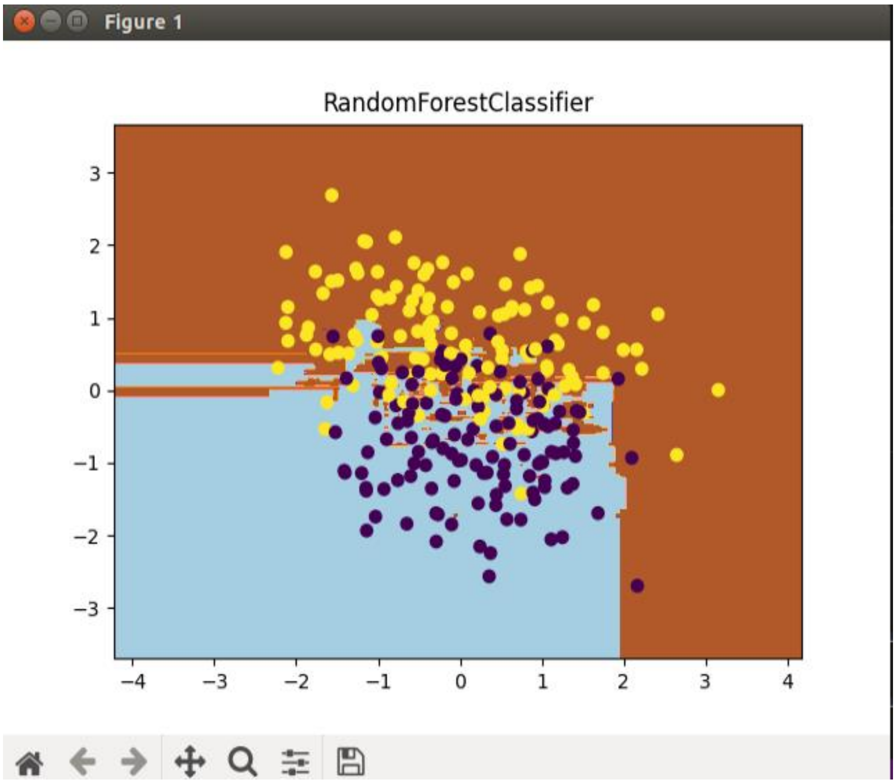


Figure 1
RandomForestClassifier

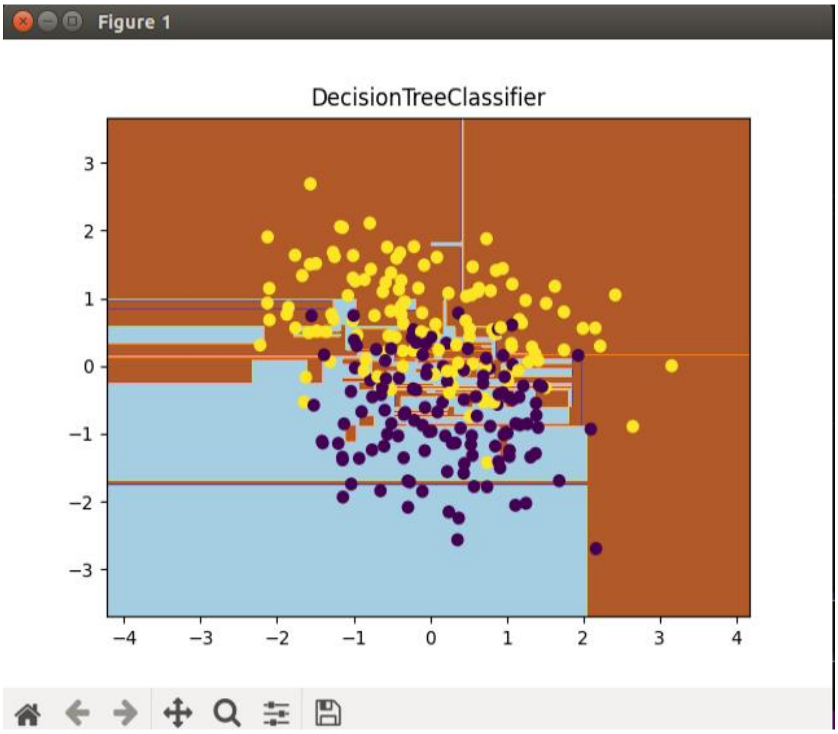


Figure 1
DecisionTreeClassifier
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

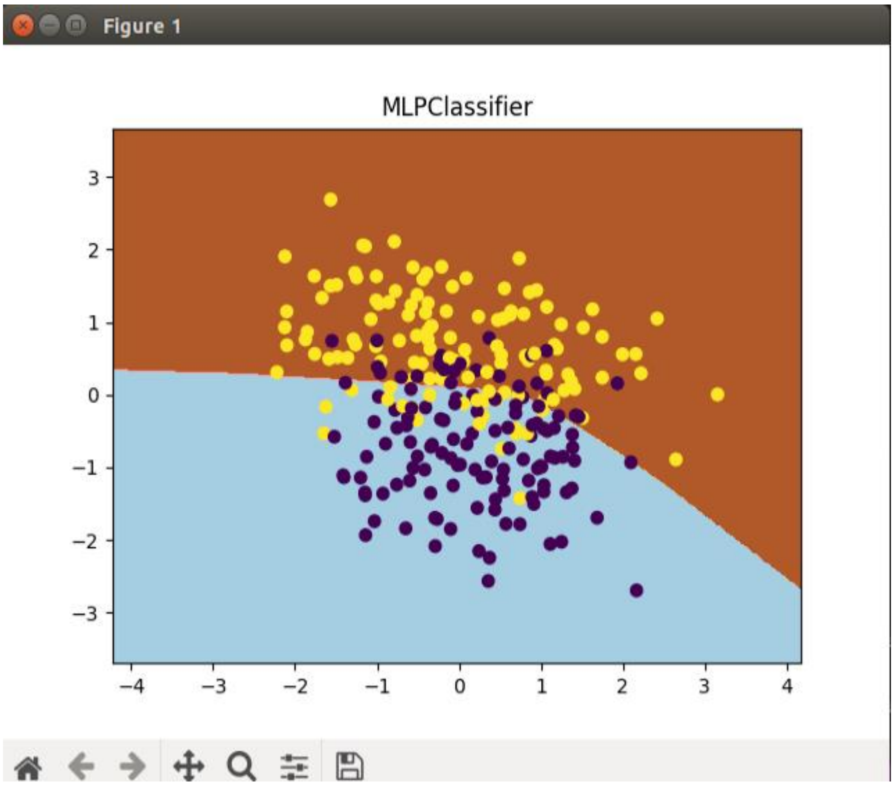
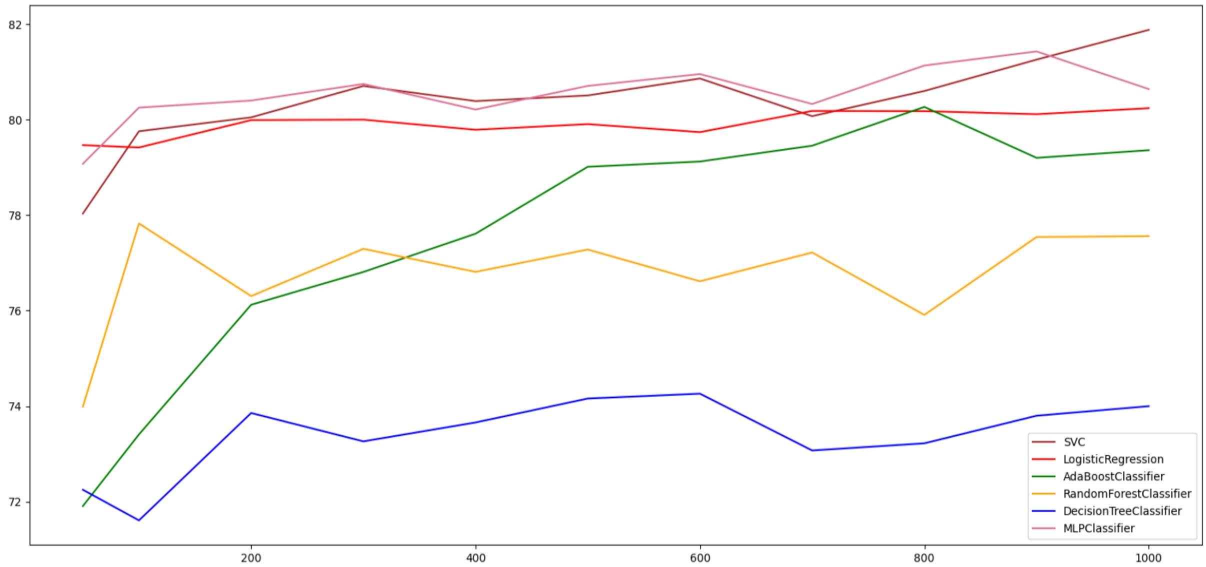


Figure 1
MLPClassifier
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

(b)

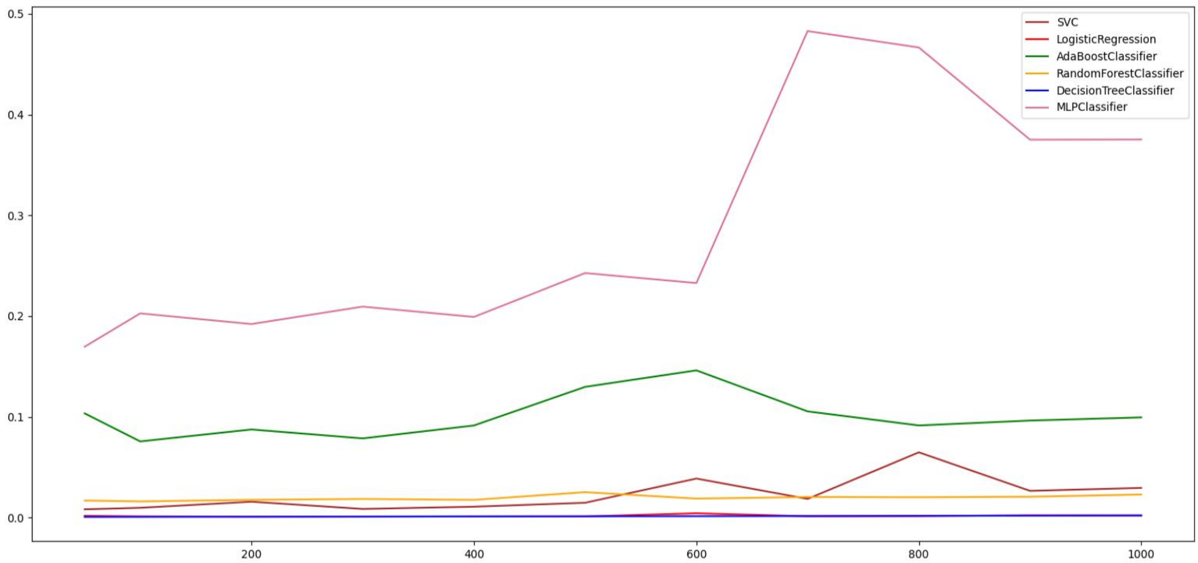
The following figure shows the average accuracy of each model, at each sample size given



What we can observe is that for the SVC, MLP and Logistic regression classifiers, regardless of the training size, the accuracy was relatively high. These methods have low variance and perform well across all sample sizes. This figure outlines the bias-variance trade-off for each method. Methods that are low variance and high bias such as SVC, Logistic Regression and Neural networks perform consistently well, whereas methods like AdaBoost decrease the variance with more samples. What we can also see is how Decision trees and Random Forest compare, Decision tree high variance, low bias, but we see the improvement of ensembling them with RF lowering variance, same trend but higher accuracy.

(c)

The following figure shows the average time taken of each model, at each sample size given



From this we can observe the following. Neural learning in general is very slow since training takes a long time, but prediction is very fast. This is one of the big drawbacks to neural learning (hard to train, complex models). We can see that for a single decision tree, it was very fast, trees are very fast learners in general and even by creating random forest of trees, the time taken to train does not drastically increase. The increased training set size for the simpler models did not have a significant increase on the training time, except for neural learning which had increased rapidly.