CSCI 5521 HW 5

Sam Johnson | Henri Parenteau | Lucas Pauling
samjohnson.ep@gmail.com | paren165@umn.edu | pauli086@umn.edu

December 10, 2020

Exercise 1

(a)

Entropy of 1 attribute: $E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$

Wait	
Yes	No
9	6

$E(Wait) = E(9,6) = E(0.6, 0.4) = 0.97$

Entropy of 2 attributes: $E(T, X) = \sum_{c \in X} P(c)E(c)$

Gain: $E(T) - E(T, X)$

		Wait		
		Yes	No	
Type	Burger	1	4	5
	Pizza	0	2	2
	Seafood	8	0	8

$E(Wait, Type) = \frac{5}{15}E(1,4) + \frac{2}{15}E(0,2) + \frac{8}{15}E(8,0) = 0.72$

		Wait		
		Yes	No	
Cost	\$	2	1	3
	\$\$	4	2	6
	\$\$\$	3	3	6

$E(Wait, Cost) = \frac{1}{5}E(2,1) + \frac{2}{5}E(4,2) + \frac{2}{5}E(3,3) = 0.9508$

		Wait		
		Yes	No	
Hunger	Yes	7	3	10
	No	2	3	5

$E(Wait, Hunger) = \frac{2}{3}E(7,3) + \frac{1}{3}E(2,3) = 0.911$

The largest gain is from Type, so we split from there. As the entropy of Pizza and Seafood is 0 for each, we set those as leaf nodes with the decision being No and Yes, respectively. The entropy of Burger is 0.72, so we further split this branch.

		Wait		
		Yes	No	
Cost	\$	0	1	1
	\$\$	1	1	2
	\$\$\$	0	2	2

$E(Wait, Cost) = \frac{1}{5}E(0,1) + \frac{2}{5}E(1,1) + \frac{2}{5}E(0,2) = 0.4$

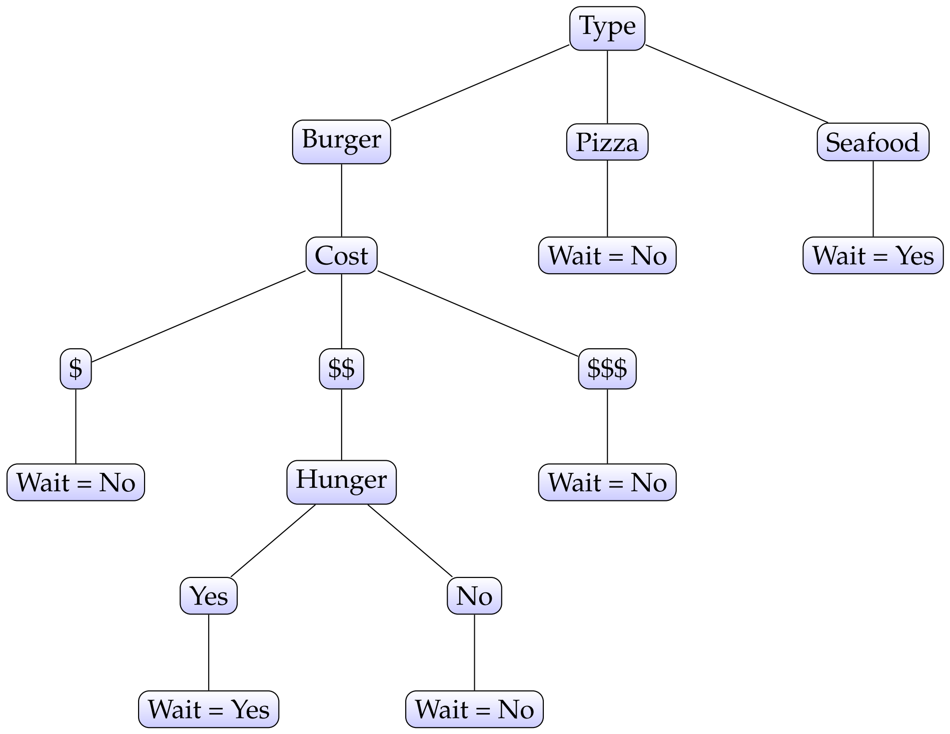
		Wait		
		Yes	No	
Hunger	Yes	1	1	2
	No	0	3	3

$E(Wait, Hunger) = \frac{2}{5}E(1,1) + \frac{3}{5}E(0,3) = 0.4$

Here the gain is the same for both, so we choose to split from Cost arbitrarily. Here both the \$ and \$\$\$ costs have 0 entropy, so those are set as leaf nodes with values No for both. The entropy of \$\$ is 0.4, so we split this branch.

		Wait		
		Yes	No	
Hunger	Yes	1	0	1
	No	0	1	1

As both values have entropy = 0, we can set the value Yes to Yes and the value No to No. Here is the final decision tree:



(b)

Seeing as the Type is Pizza, we follow that path on the decision tree and find we will not wait.

Exercise 2

Build a Perceptron [multilayer or single layer as the case may be] to recognize a certain area of the plane. That is, the Perceptron should output a "1" if the input vector lies in the shaded region.

```
a. Determine the vector of coefficients W for a single layer perceptron of the form in
Figure 1 to recognize the area in Figure 2 and again for Figure 3 shaded blue. Use a
step-function as the non-linear ‘‘sigmoid’’ activation function at designated nodes.
```

Figure 2:

In Figure 2, all points (x_1, x_2) such that $x_1 < 1$ are shaded and therefore in Class 1, all other points are in Class 2. The weight vector w must be perpendicular to the discriminant line and point towards Class 1, so we know w points in the $(-1, 0)$ direction. w also must have a bias term, so w has the form $w = (w_0, -1, 0)$. We will also append a 1 to each input point so x has the form $x = (1, x_1, x_2)$.

The activation function is a step function, so, if $x \cdot w < 0$, x will be assigned to Class 0; likewise, if $x \cdot w > 0$, x will be assigned to Class 1. Therefore we must choose the bias term w_0 such that $x \cdot w = 0$ at the discriminant boundary $x_1 = 1$.

$$\begin{aligned} x \cdot w &= 0 && (1)\\ (1,1,x_2)\cdot(w_0,-1,0) &= 0 && (2)\\ w_0 - 1 &= 0 && (3)\\ w_0 &= 1 && (4) \end{aligned}$$

Therefore $w = (1,-1,0)$ is the final weight vector for the classes in Figure 2.

Figure 3:

In Figure 3, all points (x_1, x_2) such that $x_2 > x_1 - 1$ are shaded and therefore in Class 1, all other points are in Class 2. The weight vector w must be perpendicular to the discriminant line and point towards Class 1, so we know w points in the $(-1,-1)$ direction. w also must have a bias term, so w has the form $w = (w_0, -1, 1)$. We will also append a 1 to each input point so x has the form $x = (1, x_1, x_2)$.

The activation function is a step function, so, if $x \cdot w < 0$, x will be assigned to Class 0; likewise, if $x \cdot w > 0$, x will be assigned to Class 1. Therefore we must choose the bias term w_0 such that $x \cdot w = 0$ at the discriminant boundary $x_2 = x_1 - 1$. I'll use the point $(x_1, x_2) = (1,0) \implies x = (1,1,0)$ to solve for the bias term.

$$\begin{aligned} x \cdot w &= 0 && (5)\\ (1,1,0)\cdot(w_0,-1,1) &= 0 && (6)\\ w_0 - 1 &= 0 && (7)\\ w_0 &= 1 && (8) \end{aligned}$$

Therefore $w = (1,-1,1)$ is the final weight vector for the classes in Figure 3.

```
b. Determine coefficients W and V in the 2-layer Perceptron of the form in Figure 4 to
recognize the shaded region in Figure 5.
HINT: The shaded region in Figure 5 equals the intersection of the regions of Figures 2
& 3.
```

Because Class 1 in Figure 5 is simply the intersection of Class 1 from Figures 2 & 3, we are essentially implementing the "and" function and can therefore re-use the weight vectors we calculated in part a (called w_1 and w_2).

$$W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & -1 & 1 \end{bmatrix}$$

Each point x is multiplied by the weight vectors/matrix and then evaluated by the step funciton to produce the values of the hidden nodes, z_1 and z_2. These z values will either be 0 or 1. z_1 will be 1 if the point x is in Class 1 of Figure 2; z_2 will be 1 if the point x is in Class 1 of Figure 3. Therefore points in Class 1 of Figure 5 (which is the intersection of Figures 2 & 3) will have both $z_1 = 1$ and $z_2 = 1$. We must choose V such that $(1,1)$ is assigned to Class 1, but all other possible z points ($(1,0)$, $(0,0)$, and $(0,1)$) are not.

Because the hidden layer weights vector V has a biased term, the variables have the form $z = (1, z_1, z_2)$ and $V = (V_0, V_1, V_2)$. Therefore we must choose the weight vector V such that $z \cdot V = 1$ at $(1,1)$ and 0 at all other possible points. Technically any vector pointing into the first quadrant would work, but I will chose $(1,1)$ to be the direction of V (without the bias term yet). Now I must choose the bias V_0 such that:

$$
\begin{aligned}
(1,1,1)\cdot(V_0,1,1) > 0 &\implies V_0+2>0 && (9)\\
(1,0,1)\cdot(V_0,1,1) < 0 &\implies V_0+1<0 && (10)\\
(1,1,0)\cdot(V_0,1,1) < 0 &\implies V_0+1<0 && (11)\\
(1,0,0)\cdot(V_0,1,1) < 0 &\implies V_0<0 && (12)\\
& && (13)
\end{aligned}
$$

These conditions are satisfied by any V_0 between -2 and -1, but I will choose $V_0 = -1.5$. Therefore the final weight vectors for Figure 5 are:

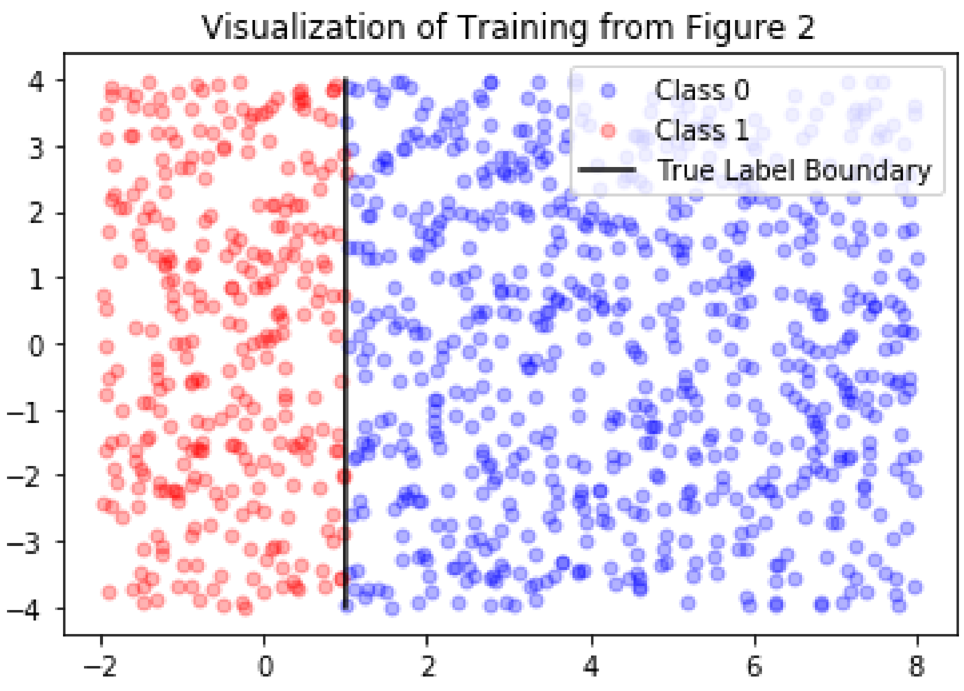
$$W = \begin{bmatrix} 1 & -1 & 0 \\ 1 & -1 & 1 \end{bmatrix} V = \begin{bmatrix} -1.5 & 1 & 1 \end{bmatrix}$$

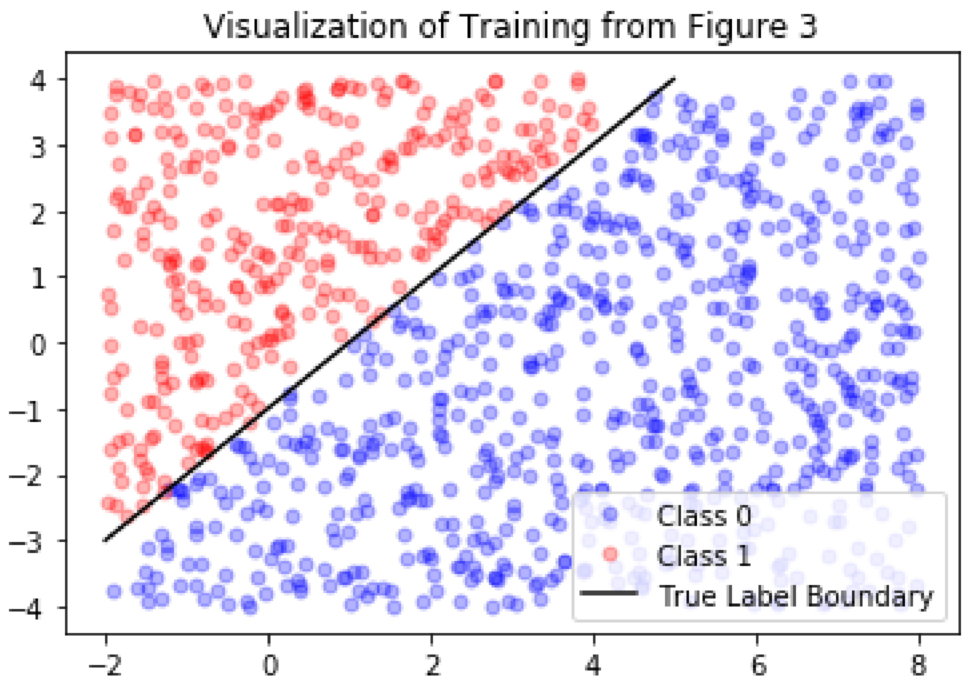
Here are some visualizations of my results:

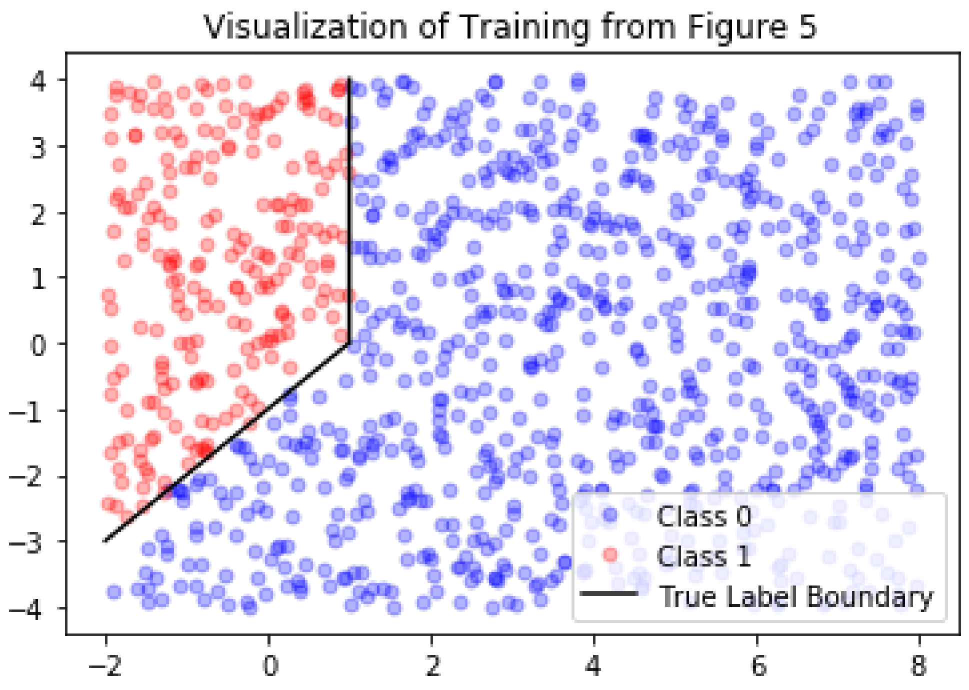
[45]:
```
import numpy as np
import matplotlib.pyplot as plt

def step(x): return np.round((np.sign(x)+1)/2.)

w1 = np.array([1,-1,0])
w2 = np.array([1,-1,1])
v = np.array([-1.5,1,1])

n = 1000
X = np.empty([n,3])
Z = np.empty([n,3])
for i in range(n):
    X[i] = np.array([1, np.random.random()*10-2, np.random.random()*8-4])
    Z[i] = [1, step(X[i].dot(w1)), step(X[i].dot(w2))]
```

[46]:
```
plt.plot(X[step(X.dot(w1))==0][:,1], X[step(X.dot(w1))==0][:,2], 'o', markersize=5,␣
 ↪color="blue", alpha=.3, label="Class 0")
plt.plot(X[step(X.dot(w1))==1][:,1], X[step(X.dot(w1))==1][:,2], 'o', markersize=5,␣
 ↪color="red", alpha=.3, label="Class 1")
plt.plot(np.ones(10000), np.arange(-4,4,8/10000.), color="black", label="True Label␣
 ↪Boundary")
plt.title("Visualization of Training from Figure 2")
plt.legend()
plt.show()

plt.plot(X[step(X.dot(w2))==0][:,1], X[step(X.dot(w2))==0][:,2], 'o', markersize=5,␣
 ↪color="blue", alpha=.3, label="Class 0")
plt.plot(X[step(X.dot(w2))==1][:,1], X[step(X.dot(w2))==1][:,2], 'o', markersize=5,␣
 ↪color="red", alpha=.3, label="Class 1")
plt.plot(np.arange(-2,5,.01), np.arange(-2,5,.01)-1, color="black", label="True Label␣
 ↪Boundary")
plt.title("Visualization of Training from Figure 3")
plt.legend()
plt.show()

plt.plot(X[step(Z.dot(v))==0][:,1], X[step(Z.dot(v))==0][:,2], 'o', markersize=5,␣
 ↪color="blue", alpha=.3, label="Class 0")
```

```
plt.plot(X[step(Z.dot(v))==1][:,1], X[step(Z.dot(v))==1][:,2], 'o', markersize=5,␣
    ↪color="red", alpha=.3, label="Class 1")
plt.plot(np.arange(-2,1,.01), np.arange(-2,1,.01)-1, color="black", label="True Label␣
    ↪Boundary")
plt.plot(np.ones(10000), np.arange(0,4,4/10000.), color="black")
plt.title("Visualization of Training from Figure 5")
plt.legend()
plt.show()
```




Visualization of Training from Figure 3
Class 0
Class 1
True Label Boundary
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8


Visualization of Training from Figure 5
Class 0
Class 1
True Label Boundary
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8

```
[ ]:
```

3.

a. Support Vectors:
   [-0.5 -0.5]
   [0.2 0.8]
   [2.0 -1.0]

b. Weight Vector: [1.0 1.0]
   Bias: 0.0

   w = c - d = [0.49982183 0.50013169] - [-0.49997676 -0.49997676]

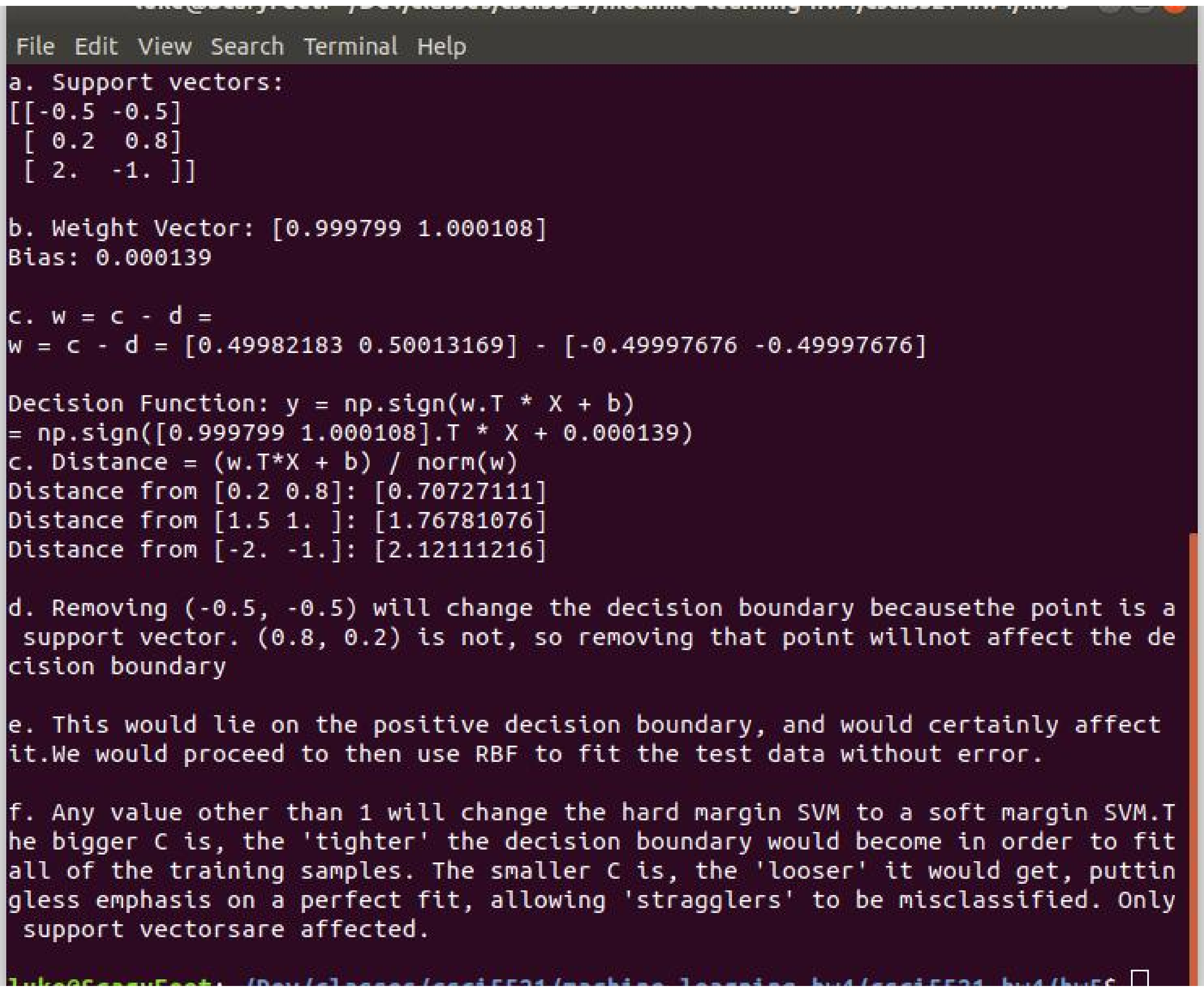
   Decision Function: y = np.sign(w.T * X + b)
   = np.sign([0.999799 1.000108].T * X + 0.000139)

   c. Distance = (w.T*X + b) / norm(w)
      Distance from [0.2 0.8]: [0.70727111]
      Distance from [1.5 1.]: [1.76781076]
      Distance from [-2. -1.]: [2.12111216]

d. Removing (-0.5, -0.5) will change the decision boundary because the point is a support vector. (0.8, 0.2) is not, so removing that point will not affect the decision boundary

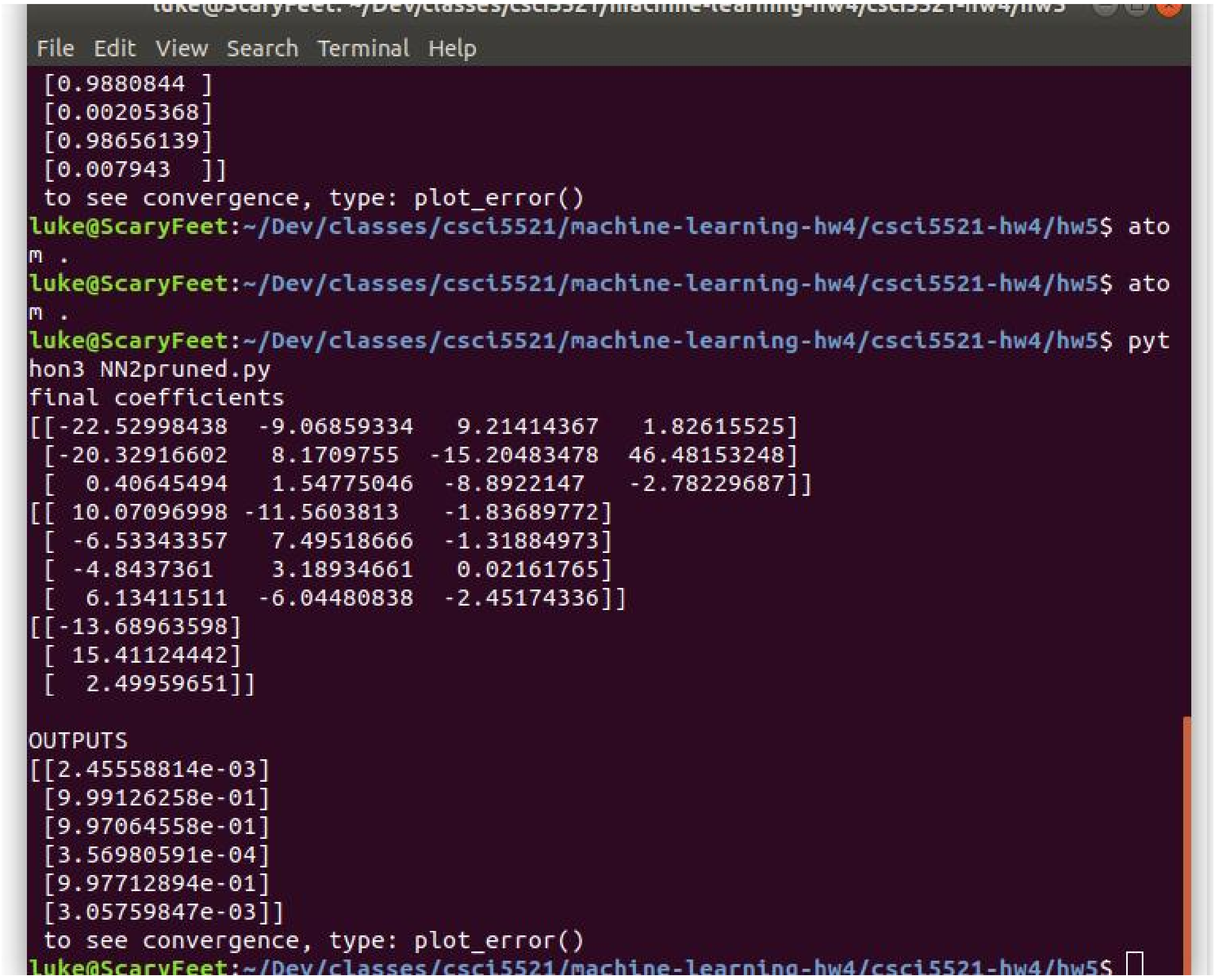
e. This would lie on the positive decision boundary, and would certainly affect it.We would proceed to then use RBF to fit the test data without error.

f. Any value other than 1 will change the hard margin SVM to a soft margin SVM.The bigger C is, the 'tighter' the decision boundary would become in order to fit all of the training samples. The smaller C is, the 'looser' it would get, putting less emphasis on a perfect fit, allowing 'stragglers' to be misclassified. Only support vectors are affected.



```
a. Support vectors:
[[-0.5 -0.5]
 [ 0.2  0.8]
 [ 2.  -1. ]]

b. Weight Vector: [0.999799 1.000108]
Bias: 0.000139

c. w = c - d =
w = c - d = [0.49982183 0.50013169] - [-0.49997676 -0.49997676]

Decision Function: y = np.sign(w.T * X + b)
= np.sign([0.999799 1.000108].T * X + 0.000139)
c. Distance = (w.T*X + b) / norm(w)
Distance from [0.2 0.8]: [0.70727111]
Distance from [1.5 1. ]: [1.76781076]
Distance from [-2. -1.]: [2.12111216]

d. Removing (-0.5, -0.5) will change the decision boundary becausethe point is a
 support vector. (0.8, 0.2) is not, so removing that point willnot affect the de
cision boundary

e. This would lie on the positive decision boundary, and would certainly affect
it.We would proceed to then use RBF to fit the test data without error.

f. Any value other than 1 will change the hard margin SVM to a soft margin SVM.T
he bigger C is, the 'tighter' the decision boundary would become in order to fit
all of the training samples. The smaller C is, the 'looser' it would get, puttin
gless emphasis on a perfect fit, allowing 'stragglers' to be misclassified. Only
 support vectorsare affected.
```

4. Answer found by running program NN2



```
 [0.9880844 ]
 [0.00205368]
 [0.98656139]
 [0.007943  ]]
 to see convergence, type: plot_error()
luke@ScaryFeet:~/Dev/classes/csci5521/machine-learning-hw4/csci5521-hw4/hw5$ ato
m .
luke@ScaryFeet:~/Dev/classes/csci5521/machine-learning-hw4/csci5521-hw4/hw5$ ato
m .
luke@ScaryFeet:~/Dev/classes/csci5521/machine-learning-hw4/csci5521-hw4/hw5$ pyt
hon3 NN2pruned.py
final coefficients
[[-22.52998438  -9.06859334   9.21414367   1.82615525]
 [-20.32916602   8.1709755  -15.20483478  46.48153248]
 [  0.40645494   1.54775046  -8.8922147   -2.78229687]]
[[ 10.07096998 -11.5603813   -1.83689772]
 [ -6.53343357   7.49518666  -1.31884973]
 [ -4.8437361    3.18934661   0.02161765]
 [  6.13411511  -6.04480838  -2.45174336]]
[[-13.68963598]
 [ 15.41124442]
 [  2.49959651]]

OUTPUTS
[[2.45558814e-03]
 [9.99126258e-01]
 [9.97064558e-01]
 [3.56980591e-04]
 [9.97712894e-01]
 [3.05759847e-03]]
 to see convergence, type: plot_error()
luke@ScaryFeet:~/Dev/classes/csci5521/machine-learning-hw4/csci5521-hw4/hw5$ 
```