Name:- Kalyan Kumar paladugula, 09th April, 2020
NetId: Kpalad4
UIN: 679025059

IML
Midterm

1) Short Answers:-

a) All of the following answers are of my own work

- p. Kalyan Kumar

b) Purpose of Regularization:

Regularization is used to reduce the complexity of a model (Regression), by penalizing the large coefficients, to avoid overfitting.

L1 Regularization:- (LASSO)

This ~~minimizes~~ regularization finds the coefficients that minimize the ~~Mean~~ Sum of squared errors plus the sum of absolute values of coefficients

$$\left(Y - \beta^T X\right) + \alpha \sum_{i=1}^{n} |\beta_i|$$

α - Regularization parameter

L2 Regularization:- (Ridge Regression)

This finds the coefficients that minimizes the both sum of squared errors and the sum of squared coefficients

$$(Y - \beta^T X) + \alpha \sum_{i=1}^{n} (B_i)^2$$

α - Regularization parameter

~~Regularization~~

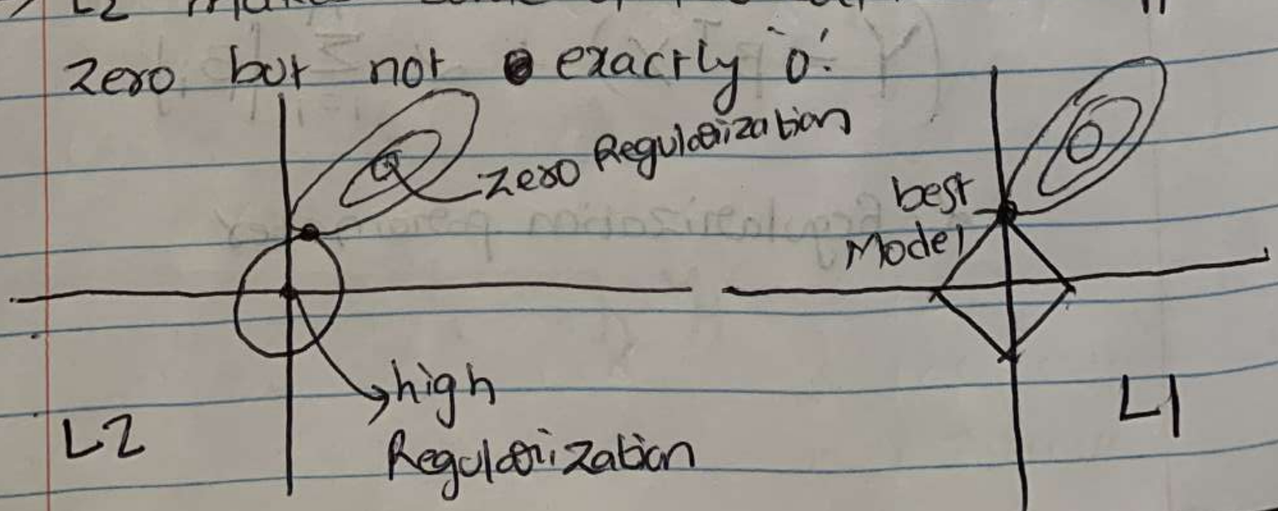
$\alpha > 0$ if α is high - underfits

α is low - overfits

Difference:-

-> L_1 makes some of the coefficients 'zero' hence used for "feature selection"

-> L_2 makes some of the coefficients approach zero but not exactly '0'.



c) Ensemble classifiers required unstable classifiers as the ~~weak~~ ~~these~~ base Learners to maintain independence

Since, the decision trees change drastically even with slight change in dataset, they are the most unstable classifiers.

Hence, the decision trees are commonly used in ensemble methods as ~~weak~~ base learners for independence.

d) ~~we~~ we use activation functions in Neural net--works to make them non-Linear. as the most of the processes in the world are non--Linear.

Neural Networks generally require activation functions ~~to do~~ for ~~supervised~~ classification and unsupervised Learning.

However, for Regression, we can used Neural Networks without an activation function

e) Yes. we can apply Kernel methods in modeling Of Neural Networks.

I would do this if the given data is not linearly seperable. I would apply the Kernel transformation on the data before feeding it to the Neural Network. So that Neural Network could easily seperate the data in higher dimensions.

Ex: I think Non-Linear SVM could be done like this.

Applying the Kernel transformed data on ~~the~~ a Neural Network decrease the training time

f) KNN is a lazy learner ie no training. So, the model doesn't have to retrain with a new data point.

If the new data point is in the margin of a SVM, the hyperplane changes but if it is not, we don't have to change the hyperplane.

But in case of Neural Networks, we have to retrain it with new data.

Hence, in the worst-case, Neural Networks take long time to retrain among NN, SVM and KNN with new data

g) <u>Bagging:</u>

Bagging trains base learners on different bootstrap samples and combines their results to form a strong learner with higher accuracy.

Generally base learners have high variance and low bias.

Bagging reduces the variance by combining/Averaging.

The variance of the final strong learner

$$Var(s) = \frac{\text{Sum of variances of base learners}}{\text{No of base learners}}$$

$$= \frac{\sum_{i=1}^{n} V_{bi}}{N}$$

Ex: consider 1000 base learners each with accuracy of 60% (binary classifiers)

So, each classifier gives the output with 60% confidence. (high variance)

If we put together the 1000 base learners, then 600 of the ~~tree~~ base learners would correctly classify and 400 would not

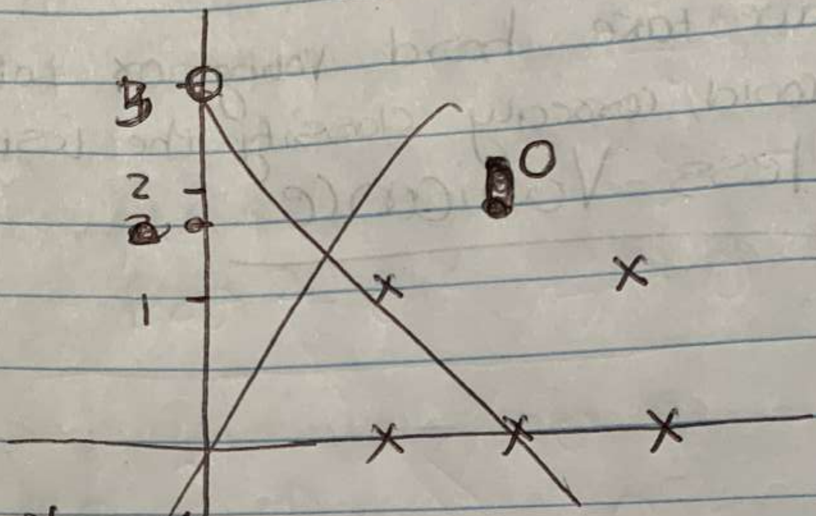
If we take hard voting or soft voting (averaging) we could correctly classify the test cases ie less Variance

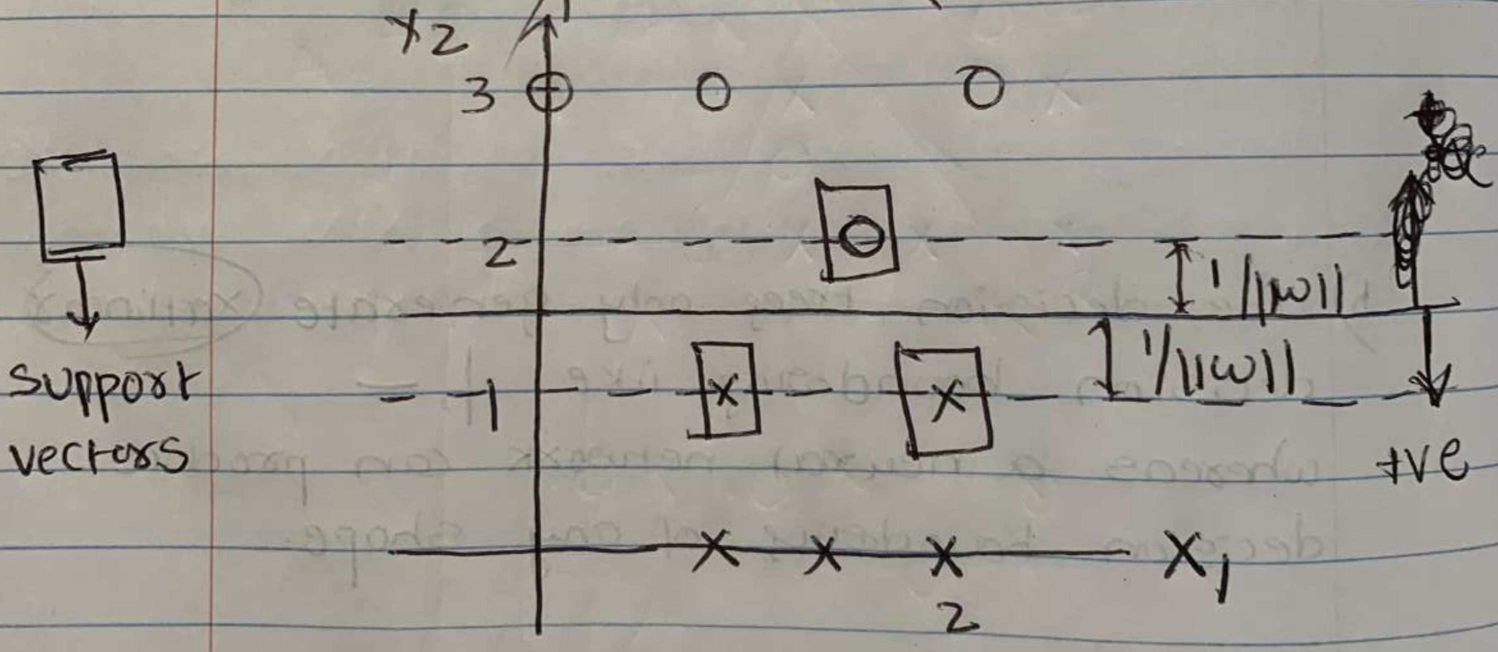
h) The decision trees only generate rectilinear decision boundaries like |, —
whereas a neural network can produce decision boundaries of any shape.

I would check for the shape of decision boundaries

2) SVM

a) Hard margin SVM:-





b) Hyperplane:-

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$0 = \beta_0 + \beta_1 \cdot 0 + \tfrac{3}{2}\beta_2 \quad - A$$

$$-1 = \beta_0 + \beta_1 \cdot 0 + 2\beta_2 . \quad - B$$

$$+1 = \beta_0 + \beta_1 \cdot 0 + \beta_2 . \quad - C$$

$B - A$

$$\Rightarrow \tfrac{1}{2}\beta_2 = -1$$

$$\beta_2 = -2.$$

$\cancel{A+C \Rightarrow}$ $\quad 2\cancel{\beta_0 + \tfrac{1}{2}\beta}$

$$B + C \Rightarrow \quad 2\beta_0 + 3\beta_2 = 0$$

$$2\beta_0 + 3(-2) = 0$$

$$\beta_0 = +3$$

From the figure

$$\Rightarrow \quad \cancel{3+} \; \beta_1 = 0$$

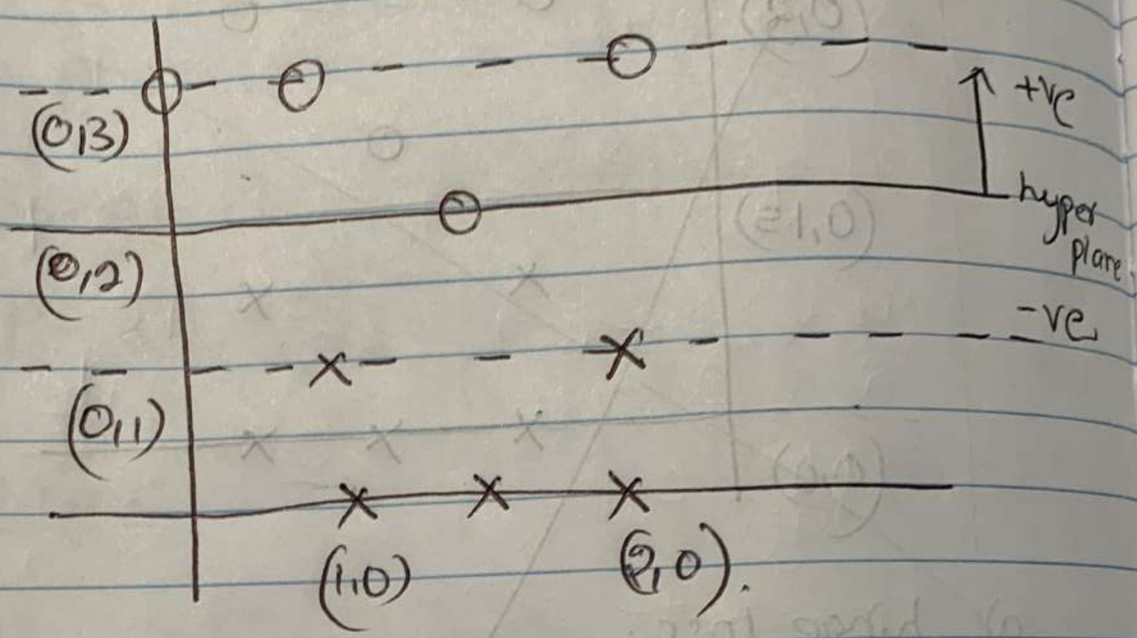
So,

$$\beta_0 = +3$$

$$\beta_1 = 0$$

$$\beta_2 = -2.$$

c) I didn't get SVM with $C = 0.1$ the closer I got is 0.2 that is.



hinge loss = 9+4+7 = 20

$0 = \beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 2 \rightarrow A$

$-1 = \beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 1 \rightarrow B$

$1 = \beta_0 + \beta_1 \cdot 0 + \beta_2 \cdot 3 \rightarrow C$

$B - C \Rightarrow -2\beta_2 = -2$

$\beta_2 = 1.$

$A \Rightarrow \quad \beta_0 + 2 \cdot 1 = 0$

$\beta_0 = -2.$

Finding c.

$$\frac{13}{2} + 0 = \frac{1+4}{2} + 20c$$

$\downarrow$ hard SVM $\qquad$ $\downarrow$ This SVM.

$$\frac{8}{2} = 20c.$$

$$c = 1/5 = 0.2.$$

d) NO, it's not the same model as before (hard margin SVM). here the hyperplane is ~~hex~~

$$y = -2 + x_2$$ → with 9 mis classifications

where as in case of hard margin svm, the hyperplane is

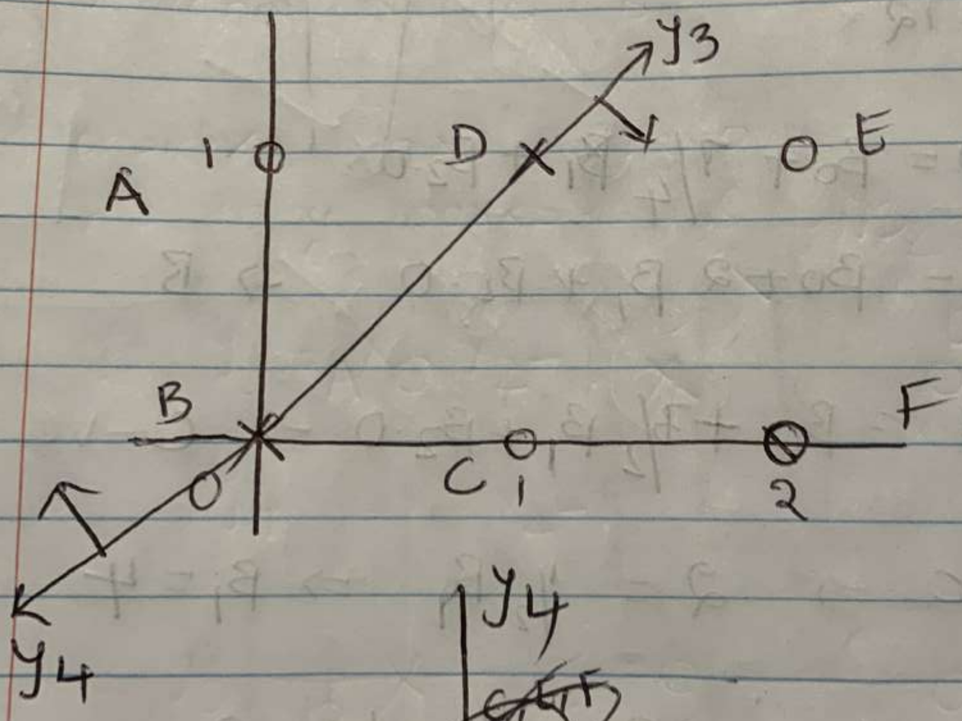
$$y = 3 - 2x_2.$$

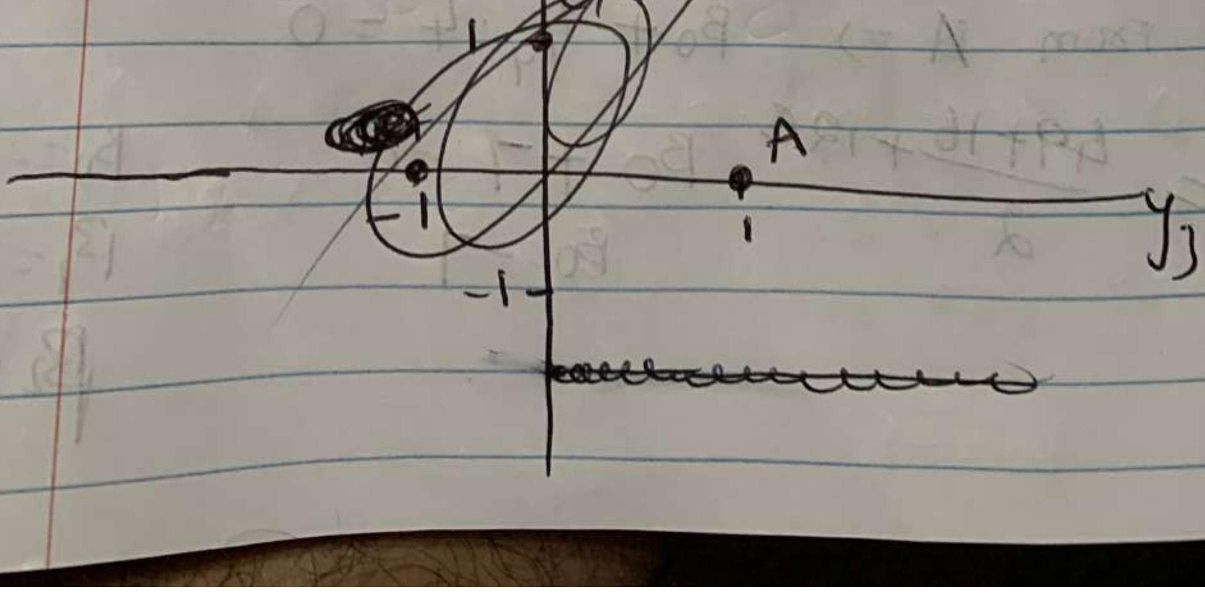
→ with '0' misclassifications

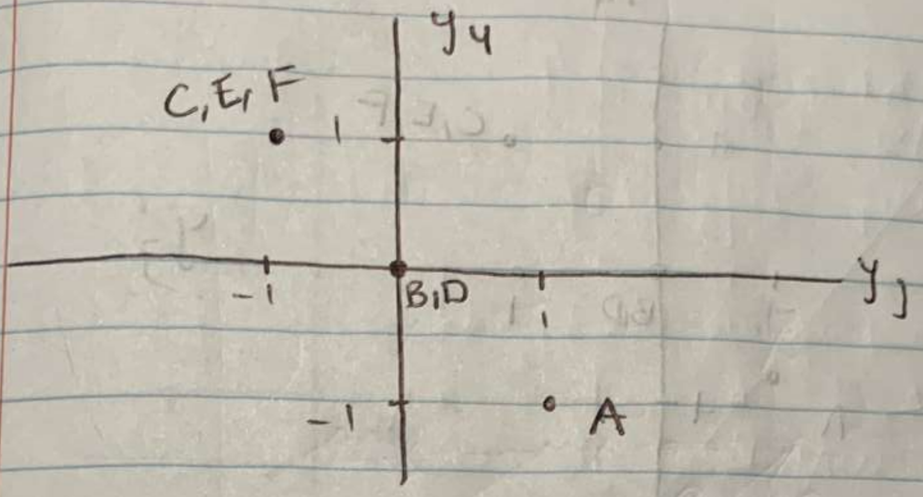
3) activation function : sign

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x>0 \\ 0 & \text{if } x=0 \\ -1 & \text{if } x<0 \end{cases}$$

(ii) positive x points are all on the boundary. consider following scenario

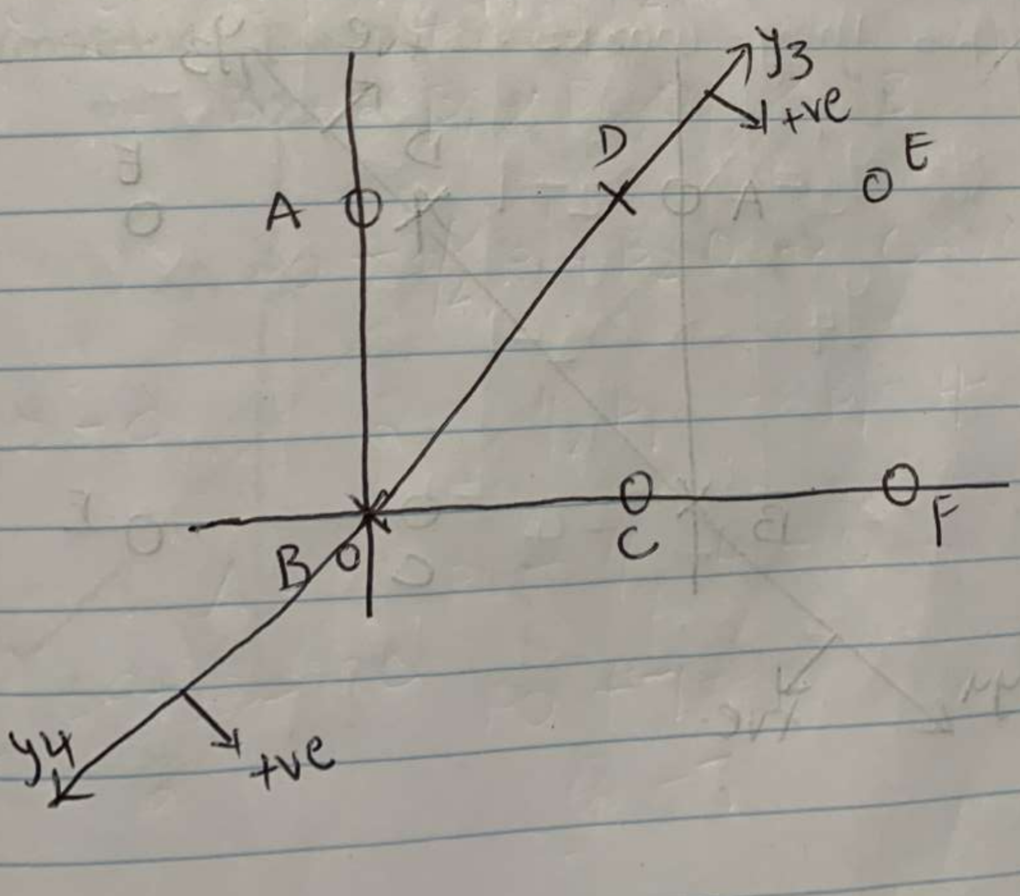


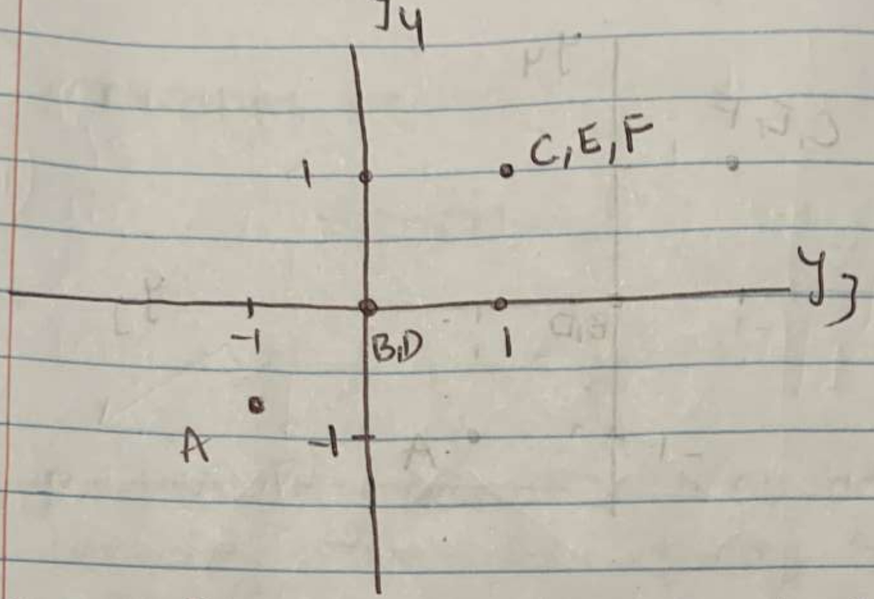




we cannot divide B, D and A, C, E, F

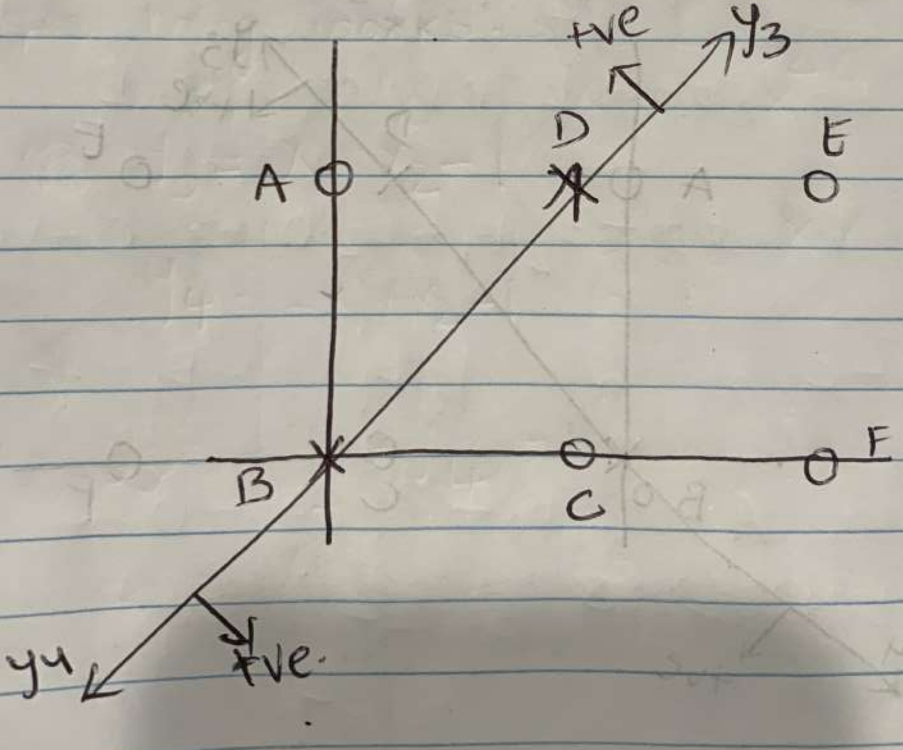
ii) consider the second scenario



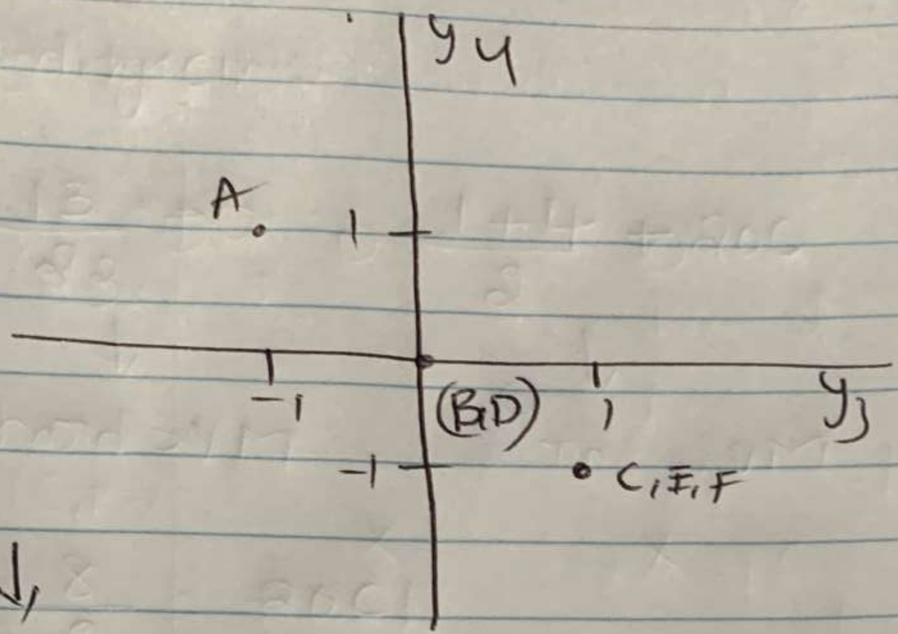


we canot divide B, D and A, C, E, F

iii) consider third scenario

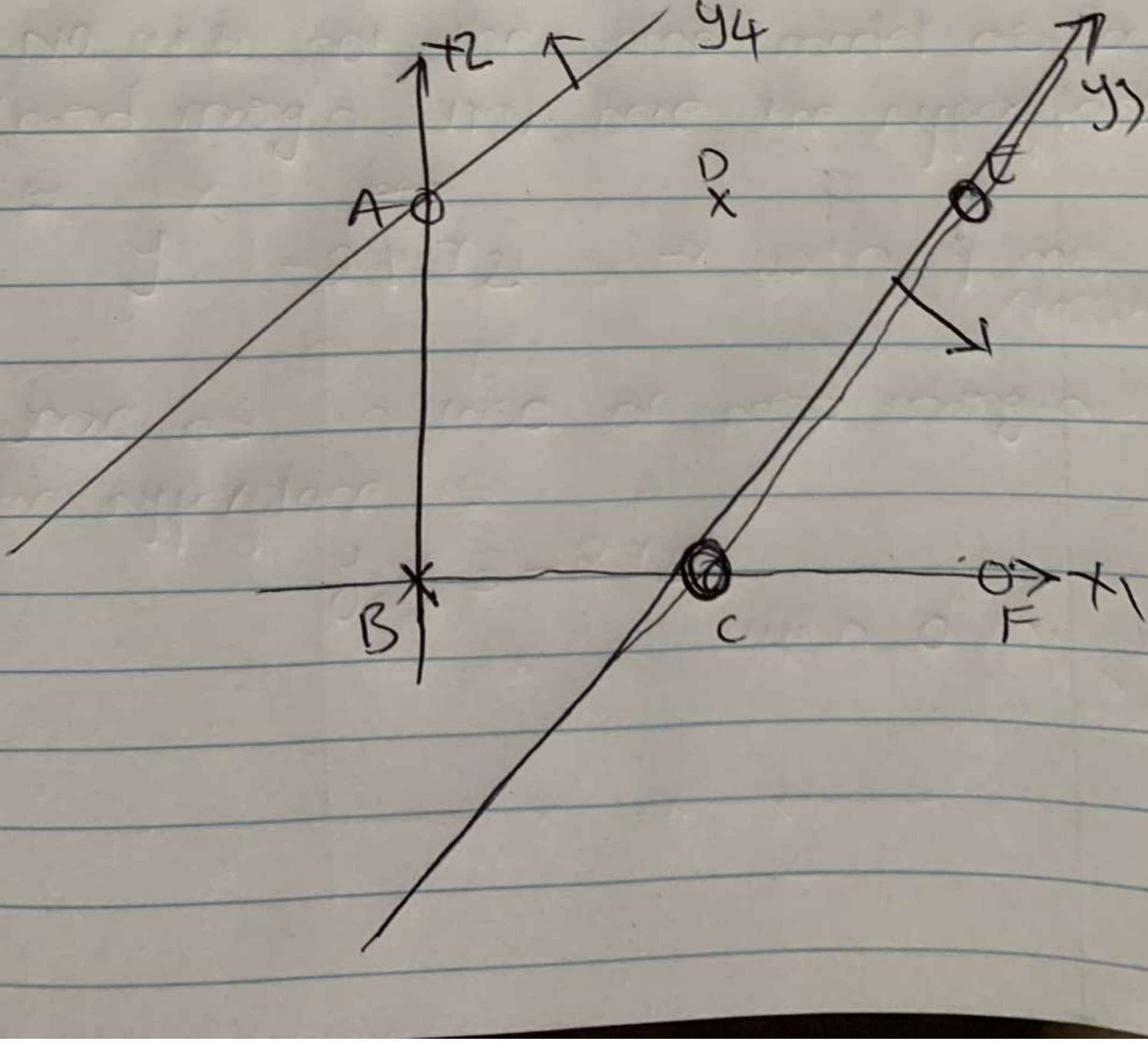


a)

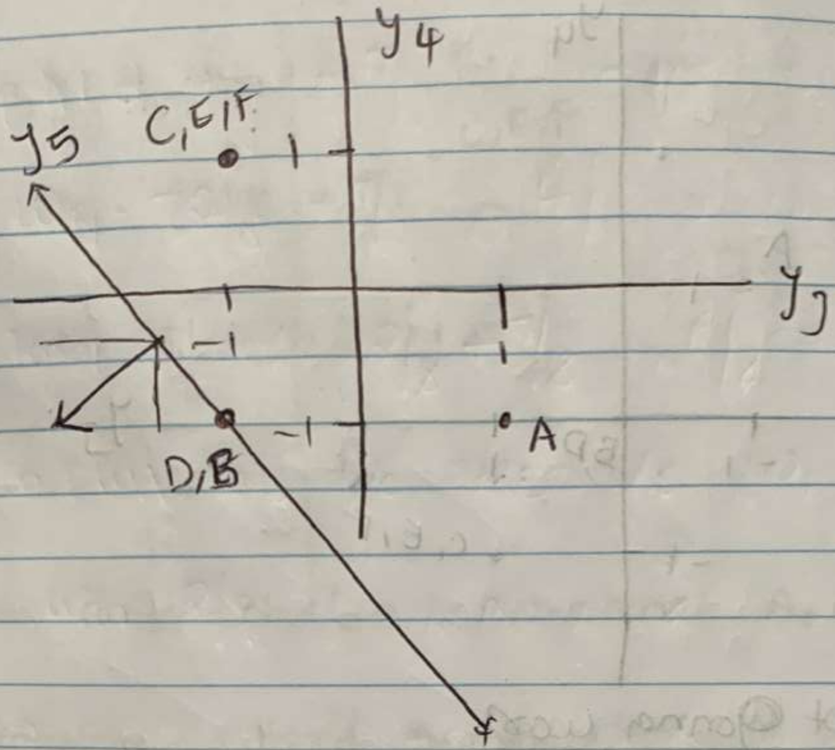


Not gonna work

a) iv) consider this fourth scenario



c, d)



This thing works.

b) $y_3 = x_1 - x_2 - 1$

$y_4 = -x_1 + x_2 - 1$

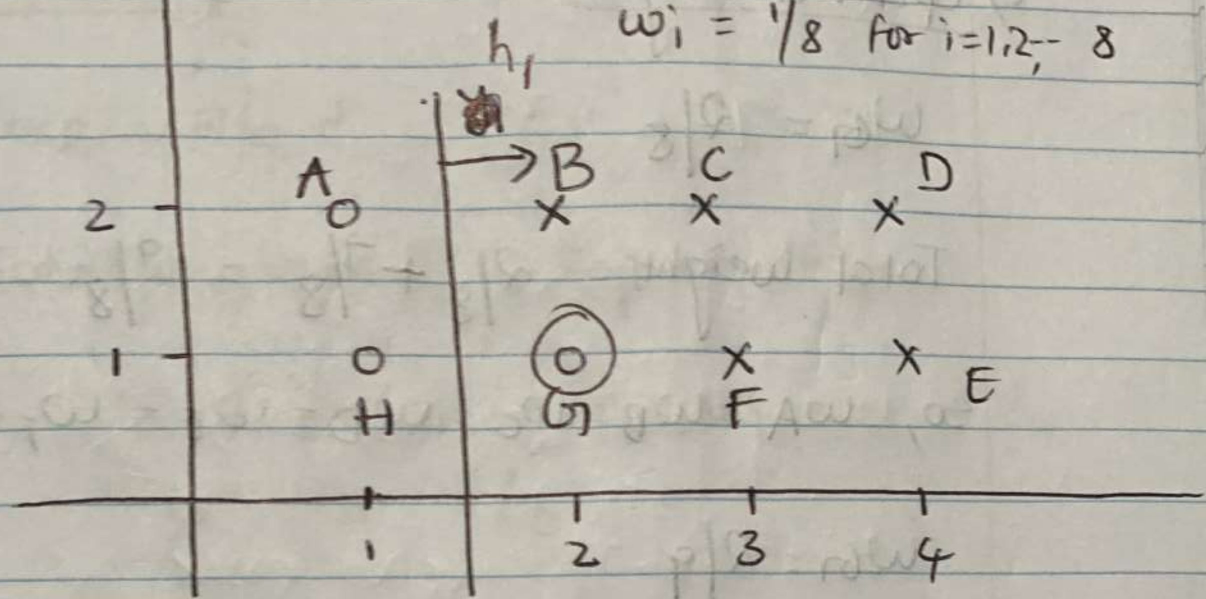
e) $y_5 = -y_3 - y_4 - 1$

Let x - positive

4)

$w_i = 1/8$ for $i = 1, 2, \ldots 8$

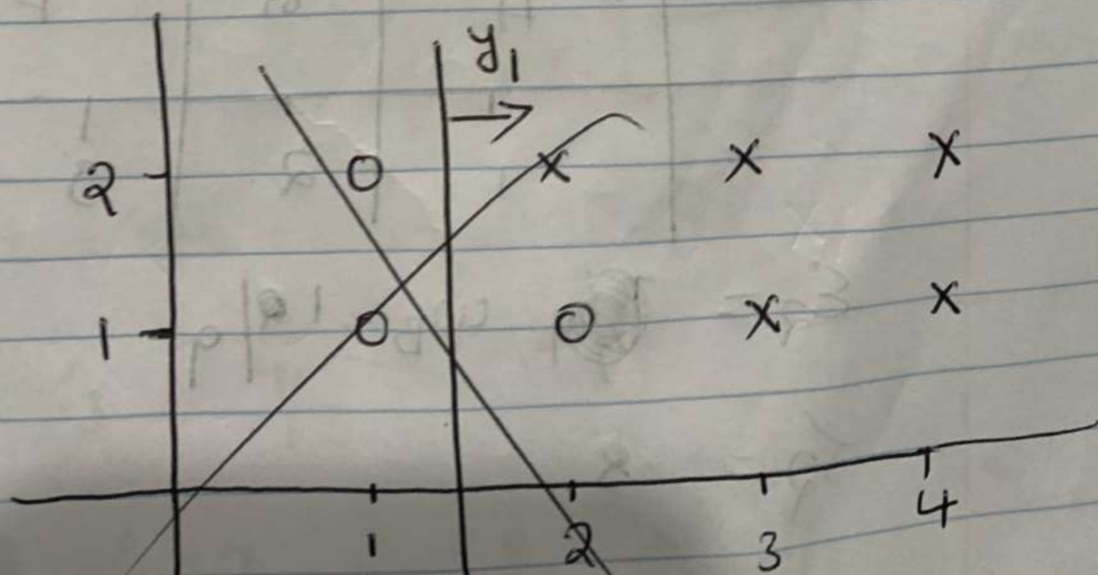
a)



a) $\text{error}_1 = 1/8 = e_1$

b) $\alpha_1 = \frac{1 - 1/8}{1/8} = 7$

c)



c) <u>updated weights</u>

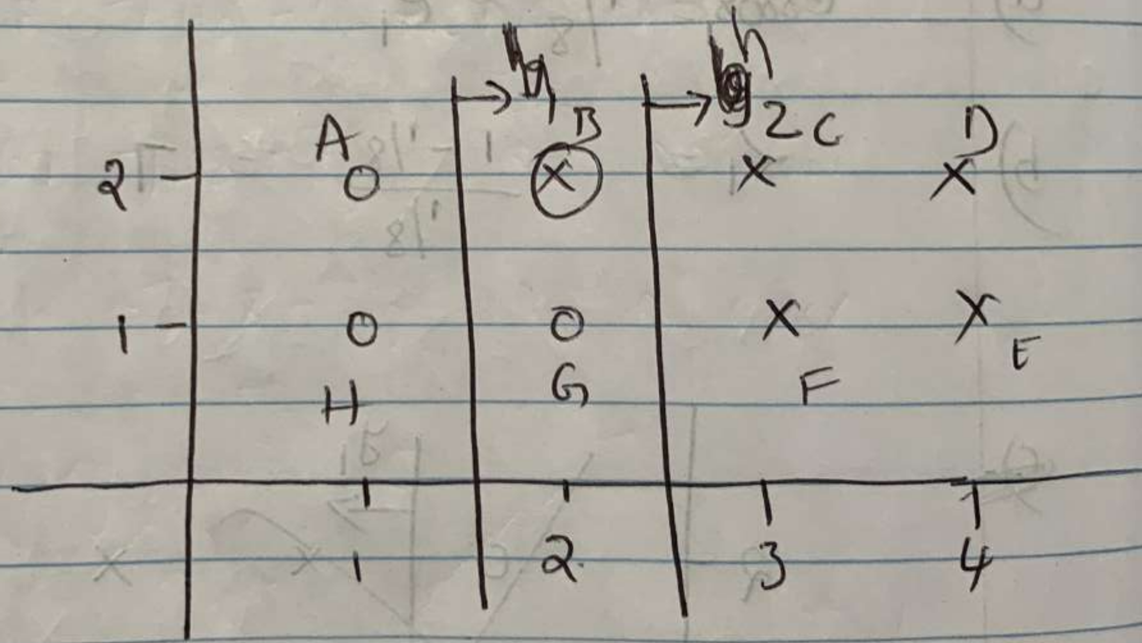
$w_G = 2/8$

Total weights $= 2/8 + 7/8 = 9/8$

so, $w_A = w_B = w_C = w_D = w_E = w_F = w_H = 1/9$

$w_G = 2/9$

so,



$\varepsilon_2 = $ [illegible]$/9$ $\quad w_B = 1$[illegible]$/9$

$\alpha_2 = 8$

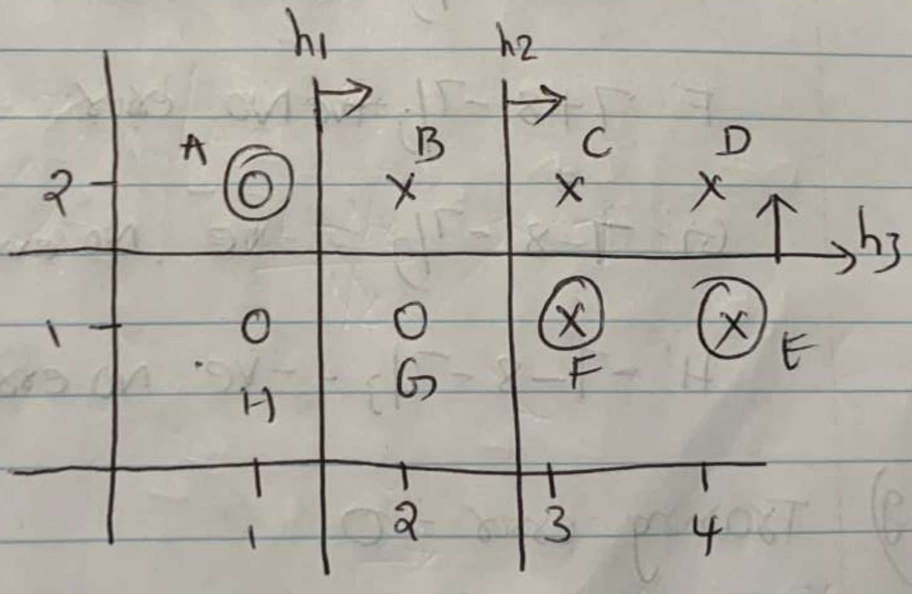
d) <u>updated weights</u>

$\omega_B = 2/9$

Total weights $= 2/9 + 8/9 = 10/9$

$1/9$

So, $\omega_A = \omega_C = \omega_D = \omega_E = \omega_F = \omega_H = 1/10$
$\omega_G = \omega_B = 2/10$



e) $\varepsilon_3 = \omega_A + \omega_F + \omega_E = 3/10$

f) $\alpha_3 = 7/3$

g) $\alpha_1 = 7, \quad \alpha_2 = 8, \quad \alpha_3 = 7/3$

A: $-7-8+7/3 = -ve$ No error

B: $7-8+7/3 = 4/3 = +ve$ No error

C: $7+8+7/3 = +ve$ No error

D: $+ve$ - No error

E: $7+8-7/3 = +ve$ No error

F: $7+8-7/3 = +ve$ No error

G: $7-8-7/3 = -ve$ No error

H: $-7-8-7/3 = -ve$ No error

g) Training error = 0

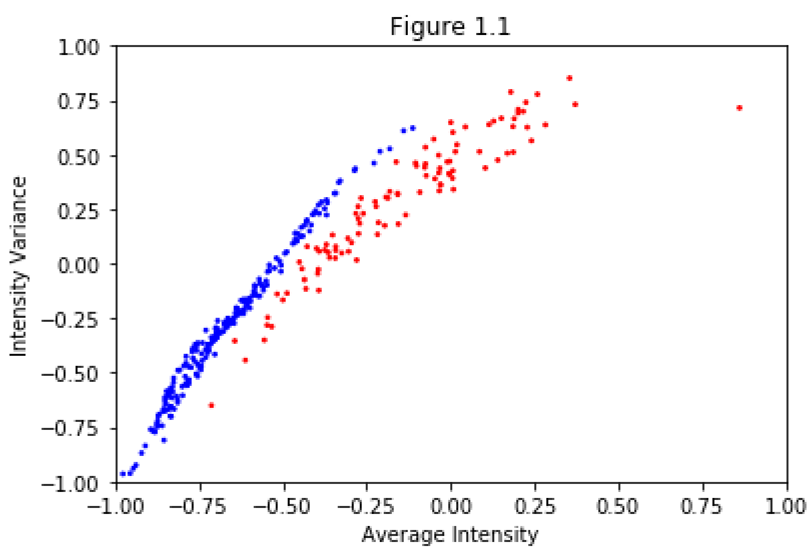
In [1]:
```
#From the console, run the following
#pip install numpy
#pip install scipy
#pip install scikit-learn
#pip install matplotlib

# Import required packages here (after they are installed)
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as mp
from pylab import show

from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.ensemble import AdaBoostClassifier


from statistics import mean, stdev, median, mode
```

In [14]:
```
def minimum(x,y):
    min = np.argmin(y)

    return x[min]
```

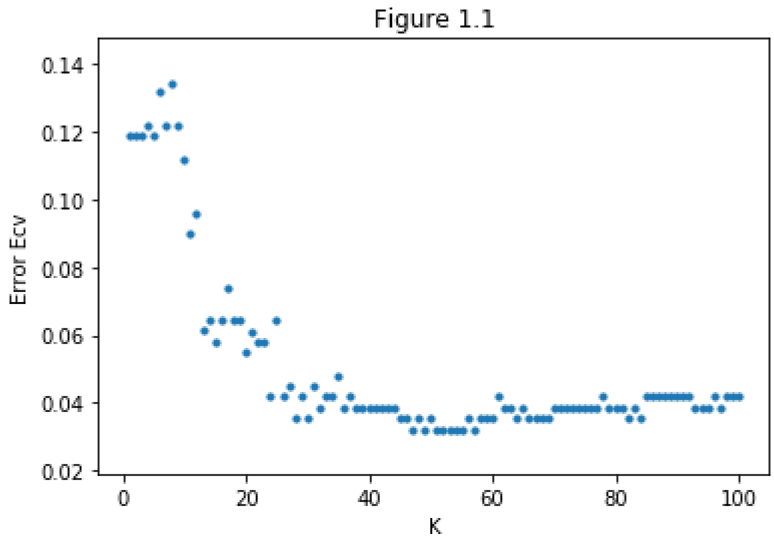
In [2]:
```
# Load data. csv file should be in the same folder as the notebook for this to
work, otherwise
# give data path.
data = np.loadtxt("data.csv")
```

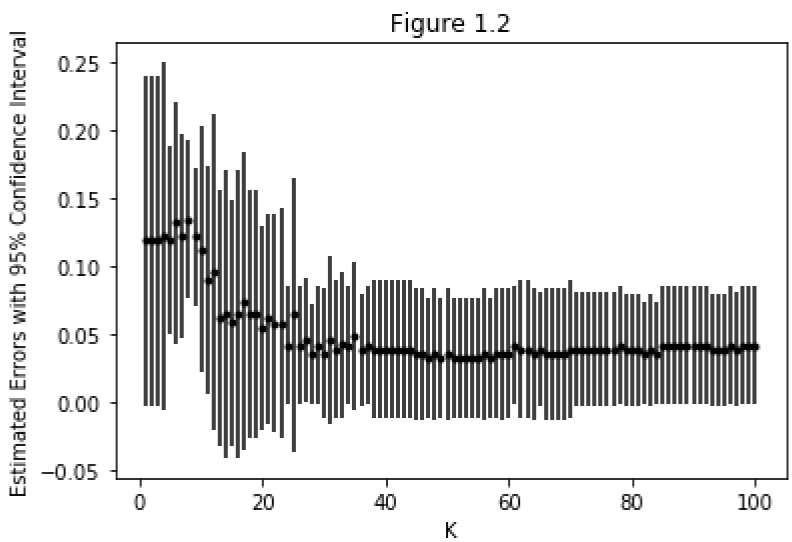
In [3]:
```
#shuffle the data and select training and test data
np.random.seed(100)
np.random.shuffle(data)


features = []
digits = []


for row in data:
    #import the data and select only the 1's and 5's
    if(row[0]==1 or row[0]==5):
        features.append(row[1:])
        digits.append(str(row[0]))


#Select the proportion of data to use for training.
#Notice that we have set aside 80% of the data for testing
numTrain = int(len(features)*.2)

trainFeatures = features[:numTrain]
testFeatures = features[numTrain:]
trainDigits = digits[:numTrain]
testDigits = digits[numTrain:]
```

In [4]:
```
#Convert the 256D data (trainFeatures) to 2D data
#We need X and Y for plotting and simpleTrain for building the model.
#They contain the same points in a different arrangement

X = []
Y = []
simpleTrain = []

#Colors will be passed to the graphing library to color the points.
#1's are blue: "b" and 5's are red: "r"
colors = []
for index in range(len(trainFeatures)):
    #produce the 2D dataset for graphing/training and scale the data so it is
 in the [-1,1] square
    xNew = 2*np.average(trainFeatures[index])+.75
    yNew = 3*np.var(trainFeatures[index])-1.5
    X.append(xNew)
    Y.append(yNew)
    simpleTrain.append([xNew,yNew])
    #trainDigits will still be the value we try to classify. Here it is the st
ring "1.0" or "5.0"
    if(trainDigits[index]=="1.0"):
        colors.append("b")
    else:
        colors.append("r")

#plot the data points
### https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html
mp.scatter(X,Y,s=3,c=colors)

#specify the axes
mp.xlim(-1,1)
mp.xlabel("Average Intensity")
mp.ylim(-1,1)
mp.ylabel("Intensity Variance")
mp.title("Figure 1.1")
#display the current graph
show()
```



In [19]:
```
# USING 2D dimensional data
x = []
y = []
z = []
p =[]
m=[]
std =[]
for i in range(1,101):
    #print(i)
    model  = AdaBoostClassifier(n_estimators = i)
    #model2.predict(testFeatures)
    cvs = cross_val_score(model, simpleTrain, trainDigits, cv = 10, scoring='a
ccuracy')
    err = 1-cvs
    evsm = err.mean()
    temp = stdev(err)
    temp2 = evsm + temp
    m.append(evsm)
    std.append(2*temp)
    p.append(temp2)
    x.append(i)
    y.append(evsm)
    z.append([x,evsm])

# print(len(x))
# print(len(y))
# print(count)
mp.scatter(x,y, s=10)
mp.xlabel("K")
mp.ylabel("Error Ecv")
mp.title("Figure 1.1")
mp.show()


mp.errorbar(x, m, yerr=std, fmt='.k');

mp.xlabel("K")

mp.ylabel("Estimated Errors with 95% Confidence Interval")
mp.title("Figure 1.2")
show()
```





In [23]:
```
b = minimum(x,m)
b
```
Out[23]: 47

Considering 95 % confident Interval

In [22]:
```
b = minimum(x,p)
b
```
Out[22]: 28

Considering error mean and 95 % confidence interval, The optimal number of estimators is 28