Section B

1a)  
165 + 34 + 12 + 96 = 307 total examples

Sensitivity = TP / TP + FN = 96/ 96 + 34 = 0.74   
Specificity = TN / TN + FP = 165/165 + 12 = 0.93

1b)

new weight = old weight + learning\_rate \*(trueLabel - output) \* input

old weight: the weight before started learning

learning rate: define how fast the output converge to a stable solution, parameter set by the user usually 0.1 (lower is more computationally expensive but will less likely miss the correct solution while higher will do bigger strides of jumps but might miss the optimal solutions a lot)

trueLabel: the actual level of training example

output: the predicted label by the perceptron

input: the features of example set

new weight: resulting weights after applying the learning rule

the learning rule is used in the learning algorithm of the perceptron:

initialise weights with random numbers between -1 and 1

for n = 1 until number of iteration

for each training example (x,y)  
 calculate activation   
 for each weight

update weight by the learning rule

end

end

end

-----

1c)

1- he can’t just correct recorded feature, the one in the range 0,50 is an outlier that still should be considered

2- overfitting is when a statistical model describes a specific dataset rather than a general case, so it describe the noise instead of underlying relationship.   
3- the perceptron convergence theorem is: if the data is linearly separable then applying the perceptron learning rule will find the decision boundary within finite number of iterations. Has nothing to do with learning rate \* space, actually where did that number come from.   
4- high accuracy doesn’t correlate to data being linearly separable, it could very well misclassified all affected students   
5- knearest isn’t a linear classifier   
K-NN rule behaved that way because he chose k = 100, which will take into account all examples, since only the minority (10) are affected, this k =100 will cause a bias toward the majority of students which is unaffected, hence why all students were classified as unaffected resulting in 90 correctly classified, this has nothing to do with it being computationally intensive

------

2)

ID3

function TreeBuilder(subsample, depth)

// base case

if(depth == 0) or (all examples in subsample have same label)  
 return the most common label in the subsample

// recursive case

for each feature in the subsample

try splitting the data

find information gained (entropy)

end for

choose feature that has the maximum gain max(entropy before split - entropy post split)

split subsample into left/right subsample

left tree ← TreeBuilder(leftSubSample, depth -1)

right tree ← TreeBuilder(rigghtSubSample, depth -1)

return tree

end function

b)

Filter method: select variable regardless of the model, suppressing the least interesting variable. operating only on the correlation between a variable and predict.

Wrapper method: Evaluate subset of variables, unlike filter method it detect possible interaction between variables.

Forward selection:  
start with no variable in model  
test the addition of each new variable against the chosen model

adding variable if that improve the model

repeat until no more improvements are made

c)

Ensemble: fits multiple models to the training data, called the base learner, and when clarification is required, it takes a committee voting to vote on the data.

Boosting and Bagging

Bagging:

A parallel algorithm each model works independently from the other

Function Bagging(input trainingData + labels, number of model M)

for j = 1 to M

take bootstrap sample T’ from T

Build a model on T’

Add model to the set

end for

return the set of model, for a testing point x get the response from majority vote

**N.B** Bootsrap:

Generating multiple dataset from original data set, by selecting n training example from N with replacement.

Boosting:

1- take a bootstrap of the dataset

2- Train one model on the bootstrap

3- Take a look which examples the model got wrong

4- Upweight the hard examples, downweight the easy ones

5- return to step 2 with updated weight until get the committee vote needed

Boosting is serial/dependent on previous result.

Bagging: works in parallel

Boosting: is serial

-------------

Section C

3-

1. lol nope
2. b) lol nope\_

4-

a)

Ignore images they are incorrect, you are supposed to only od this:

- compute the centroid points

- Get the distance from each point to the centroid in each cluster, square it

- sum all points in each cluster

- sum all clusters

only in each X in each cluster

[picture](https://drive.google.com/open?id=1T2wYkwMsvo2aoOMHQA6o30X84OcpkBJNBg) [picture2](https://drive.google.com/open?id=1_I2Yshfc5qg0wLIAuTeqgIPN8bNZjrcF4g)

b) k-means

given cluster number K

Choose seed points initially from the data

1- assign each data object to the cluster with nearest seed point

2- compute new seed point (centroid) from the mean points of the clusters

3- go back to step 1 until no more assignments are done.

the need to know number of clusters K in advance, noisy outlier might mess up the data as well

c)

X1: K-mean: since no labels are provider an unsupervised learning algorithm is needed

X2: K-nearest: since labels are provided knearest would have better accuracy. a supervised learning algorithm can be used

Distance algorithm:

X1: use internal index to decide on best partition using different distance measures   
X2: use cross validation with different distance measures until find optimal value of k

X1: Minkowskai, with p found using trial and error

X2: Euclidean, based on various tests done by experts on wikipedia… using cross validation