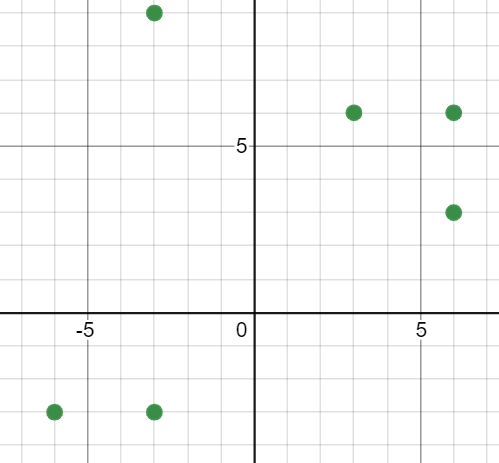
Yes

q91.a)

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Cluster center point | Points belonging to cluster | New cluster center |
| Step 1 | Cluster 1 : (3,6) | A,B,C | (5,5) |
| Cluster 2 : (-3, 9) | D,E,F | (-4,1) |
| Step 2 | Cluster 1 : (5, 5) | A,B,C | (5,5) -> stable |
| Cluster 2 : (-4, 1) | D,E,F | (-4, 1) -> stable |



1.b) In order to determine the optimal number of clusters for the points, we could run the algorithm with several values of k, plot the score for each value of k, which should yield a decreasing convex curve. By using the elbow method, we could look at the point of inflection of the curve, which is the point where the model fits the data the best without being too complex.

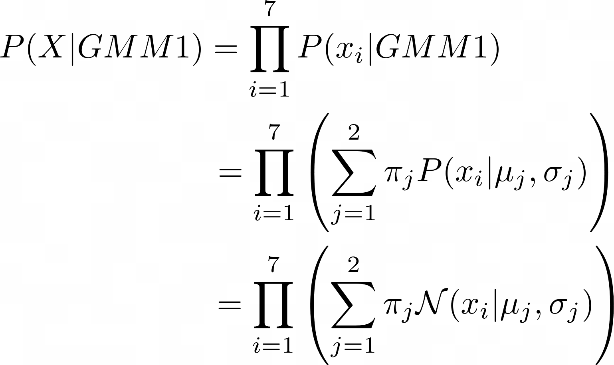
1.c) This question is very limited since we would need the true values of the normal distribution around 0 (rounding it to 0 yields errors because log(0)).

(Can we maximise the likelihood instead of minimize log likelihood? Deals with problems of having 0s, and the motivation for having log is to deal with tiny values anyways and isn’t mathematically relevant?)

Theoretically : (work in progress)

We will write X = (x1, x2, …, x7) = (1, 2, 3, 5, 8, 10, 11)

For GMM1 :

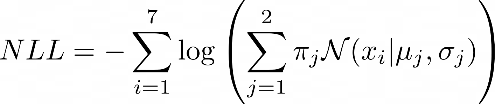


P(X | GMM1) = \prod\limits\_{i=1}^7 P(x\_i | GMM1) 78

= \prod\limits\_{i=1}^7 \left( \sum\limits\_{j=1}^2 \pi\_j P(x\_i | \mu\_j, \sigma\_j ) \right)

= \prod\limits\_{i=1}^7 \left( \sum\limits\_{j=1}^2 \pi\_j \mathcal{N}(x\_i | \mu\_j, \sigma\_j ) \right)

Similarly for GMM2.

You could compare the negative log-ins likelihoods tead, which would give (I think) : 

NLL(GMM1) = 67.81 < NLL(GMM2) = 150.25, so GMM1 is better.

1.d) Explain GMM-EM algorithm (see lecture on EM)

t

2.a)

Final decision tree :

Review

Bad | Good

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

| |

NO Price is expensive ?

|

Yes | No

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

| |

NO YES

How to find it :

1st step : compute the best information gain for all possible splits:

H(dataset) = sum –p\_i log2(p\_i) = -0.6 log2(0.6) -0.4 log2(0.4) = 0.97095059445

* Quality : H(DS|Q=H) = 0.918
* Quality : H(DS|Q=S) = 0.918
* Quality : H(DS|Q=P) = 1
* Brand : H(DS|B=F) = 0.918
* Brand : H(DS|B=M) = 1
* Brand : H(DS|B=U) = 0
* IG(DS, Brand) = H(DS) - 6/10 \* H(DS|B=Mainstream) - 3/10 \* H(DS|Brand=Famous) - 1/10 \* H (DS|Brand= Unknown) = 0.0946
* Price : IG(P) = 0.2805
* Review : IG(R) = 0.4192

Biggest IG is that of Review, so split on Review. Left child has 0 entropy, so must be a leaf node, assign label. Right child:

H(DS|Good) = 0.918

Instead of iterating through every category, notice every value of Price leads to 0 entropy.

H(DS|Good, Expensive) = H(DS|Good, Cheap) = 0

So split on Price. Entropy is now 0 for the two children nodes, so they must be leaves, assign labels.

Every node is a leaf, stop algorithm.

**Alternative full working:**

Calculating split for D:

H(D): 6/10 \* log2(6/10) + 4/10 \* log2(4/10) = 0.9709505944546686

H(Quality\_High): 2/3 \* log2(2/3) + 1/3 \* log2(1/3) = 0.9182958340544896

H(Quality\_Poor): 3/4 \* log2(3/4) + 1/4 \* log2(1/4) = 0.8112781244591328

H(Quality\_Standard): 1/3 \* log2(1/3) + 2/3 \* log2(2/3) = 0.9182958340544896

H(Quality): (3/10 \* 0.9182958340544896) + (4/10 \* 0.8112781244591328) + (3/10 \* 0.9182958340544896) = 0.875488750216347

IG(D, Quality) = 0.0

442383216

H(Brand\_Famous): 2/3 \* log2(2/3) + 1/3 \* log2(1/3) = 0.9182958340544896

H(Brand\_Mainstream): 3/6 \* log2(3/6) + 3/6 \* log2(3/6) = 1.0

H(Brand\_Unknown): 1/1 \* log2(1/1) = 0.0

H(Brand): (3/10 \* 0.9182958340544896) + (6/10 \* 1.0) + (1/10 \* 0.0) = 0.8754887502163469

IG(D, Brand) = 0.09546184423832171

H(Price\_Cheap): 3/7 \* log2(3/7) + 4/7 \* log2(4/7) = 0.9852281360342515

H(Price\_Expensive): 3/3 \* log2(3/3) = 0.0

H(Price): (7/10 \* 0.9852281360342515) + (3/10 \* 0.0) = 0.6896596952239761

IG(D, Price) = 0.2812908992306925

H(Review\_Bad): 4/4 \* log2(4/4) = 0.0

H(Review\_Good): 2/6 \* log2(2/6) + 4/6 \* log2(4/6) = 0.9182958340544896

H(Review): (4/10 \* 0.0) + (6/10 \* 0.9182958340544896) = 0.5509775004326938

IG(D, Review) = 0.4199730940219748

Split on Review

D\_1 (Bad): Rows [1 2 7 9] - All no

D\_2 (Good): Rows [ 3 4 5 6 8 10]

Calculating split for D\_2:

H(D\_2): 2/6 \* log2(2/6) + 4/6 \* log2(4/6) = 0.9182958340544896

H(Quality\_High): 1/2 \* log2(1/2) + 1/2 \* log2(1/2) = 1.0

H(Quality\_Poor): 1/1 \* log2(1/1) = 0.0

H(Quality\_Standard): 1/3 \* log2(1/3) + 2/3 \* log2(2/3) = 0.9182958340544896

H(Quality): (2/6 \* 1.0) + (1/6 \* 0.0) + (3/6 \* 0.9182958340544896) = 0.792481250360578

IG(D\_2, Quality) = 0.12581458369391152

H(Brand\_Mainstream): 1/4 \* log2(1/4) + 3/4 \* log2(3/4) = 0.8112781244591328

H(Brand\_Unknown): 1/1 \* log2(1/1) = 0.0

H(Brand\_Famous): 1/1 \* log2(1/1) = 0.0

H(Brand): (4/6 \* 0.8112781244591328) + (1/6 \* 0.0) + (1/6 \* 0.0) = 0.5408520829727552

IG(D\_2, Brand) = 0.37744375108173434

H(Price\_Expensive): 2/2 \* log2(2/2) = 0.0

H(Price\_Cheap): 4/4 \* log2(4/4) = 0.0

H(Price): (2/6 \* 0.0) + (4/6 \* 0.0) = 0.0

IG(D\_2, Price) = 0.9182958340544896

Split on Price

D\_2\_1 (Expensive): Rows [3 6] - All No

D\_2\_2 (Cheap): Rows [ 4 5 8 10] - All Yes

2.b) Genotype: a binary vector of size 18

Phenotype: our neural network with the given hyperparams in accordance with the following function:

* First 2 bits: number of hidden layers (1,2,3,4)
* Next 3 bits: number of neurons in layer 1 (1,3,5,10,15,20,30,50)
* Next 3 bits: number of neurons in layer 2 if it exists (1,3,5,10,15,20,30,50)
* Next 3 bits: number of neurons in layer 3 if it exists (1,3,5,10,15,20,30,50)
* Next 3 bits: number of neurons in layer 4 if it exists (1,3,5,10,15,20,30,50)
* Next 2 bits: Activation function (tanh, sigmoid, relu, linear)
* Last 2 bits: Learning rate (1, 0.5, 0.1, 0.01)

// Extra bits to control mutation rate? That would require having a predetermined list of them and choosing from that list +1

2.c) Genetic operators:

* Selection: Any method of selecting the best individuals given their fitness function, for example, tournament selection or proportionate selection, roulette wheel etc. Also pass the best solutions straight to next generation in order to ensure that the best fitness per generation never decreases.
* Crossovers: from two parents, generate possible children by splitting at a position in the binary vector (either 2, 5, 8, 11, 14, 16 -> between two “genes”)
* Mutation: probability that a random bit will flip 0 <-> 1

2.d) Fitness function : We need a way to evaluate the individual (which is a classifier network), so we can use any appropriate metric : average classification rate, F1 scores, etc. Evaluate final network performance using cross validation over the entire dataset. (describe cross validation steps) The performance Is given by the average performance over all the folds.

3.a) The neural net seems to be under-fitting: we expect models to be able to reach error rates of around 10%, but this one has about 40%. It does not seem able to learn more than this, thus, we should change its structure (number of hidden layers, number of neurons per layer, activation function) in order for it to become more complex and become able to learn more and fit our data better. This can also be due to the learning rate being so small that the optimization gets stuck in a local minimum, so increasing the learning rate might help. In the case of it underfitting, introducing learning rate decay can also help, as it could be the case that the network is “bouncing around” in a bowl containing the global minimum but can’t reach it if we don’t decay the learning rate introducing. There might also not be enough training data.

3.b) This question asks about material that has not been covered this year. The vanishing gradient problem happens in networks with several hidden layers, where the derivative of the loss wrt the weights of the nth layer is normal in the last layers but becomes increasingly small as n decreases (as you backpropagate the error). This causes a difficulty for the network to train its first layers. It can be prevented using activation functions like ReLU, which has no “maximum” and thus never reaches a state where it saturates.

3.c) multi class classification problem with 4 classes: 4 output neurons, softmax activation so that sum of neurons is 1 and the nth neuron’s value represents the probability that it belongs to class n.

Appropriate loss function: -> categorical cross-entropy

3.d)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | P R E D I C T E D | | | |
| T  R  U  E | 6 | 2 | 1 | 1 |
| 1 | 7 | 1 | 1 |
| 3 | 2 | 0 | 5 |
| 1 | 1 | 2 | 6 |

Global accuracy : (6+7+0+6)/40 = 19/40

Class 1 : recall=6/10, precision=6/11, F1=0.571

Class 2 : recall=7/10, precision=7/12, F1=0.636

Class 3 : recall=0, precision=0, F1=not defined (0 by convention)

Class 4 : recall=6/10, precision=6/13, F1=0.522

UAR = Unweighted Average Recall = (6/10+7/10+0+6/10)/4=19/40

(I think we called it macro averaged recall somewhere (slide 41), it’s been explained in a Piazza post)

NB : here it is equal to global accuracy because there are the same number of examples per class, I think?

Comment : the neural net is not good for classifying class 3, which affects negatively the precision of other classes. Ideas on how to solve it : increase training on class 3 with more examples for instance

3.e) It just asks whether the given p value is significant. I think it is not since > 0.05, and we can increase the amount of test data to check for better results.

Maybe you could fix the seed etc to reduce noise