CS 760 - Homework Assignment #5

Part 1 - Learning Theory

Answer the following questions regarding VC dimension and PAC learning. Show all of your work explicitly for each solution.

1. Suppose your hypothesis class (H) consists of decision trees with 7 nodes that split on only one feature. Calculate the VC dimension of H.

ANSWER:

Let us consider a continuous feature (f) on which the decision tree splits the instances on. Since there are 7 nodes, this means there are 7 threshold values for the feature found by the decision tree algorithm. Thus, if we consider a number line to represent the values of the feature (and the corresponding instances), the 7 thresholds would split the number line at 7 places and forming 8 segments (or 8 ranges of values and labels for each range as indicated by the decision tree). For example, if the splits are represented as the following table

The VC dimension of hypothesis space H is the size of the largest set of instances that is shattered by H. In the given case, the hypothesis space is the set of all decision trees that perform splits at 7 different positions in the number line for the feature. Thus, if there are eight instances (with eight values for the feature f), the hypothesis can be made to choose thresholds / splits accordingly and can label each of the eight ranges with the true class of the instance. If there are more than 8 instances then multiple instances falling under the same range may have different class labels and cannot be shattered.

Thus, the size of the largest set of instances that can be shattered by H is 8, hence the VC-dimension is 8. **VC-dim(H) = 8**

2. Use this VC dimension to derive the sample complexity of H with $\varepsilon = 0.05$ and $\delta = 0.01$. In words, describe what ε and δ represent, as well as what the sample complexity tells you about H with respect to ε and δ.

ANSWER:

The sample complexity for the hypothesis is given as follows,

$$m >= 1/\varepsilon\ (4.\log_2(2/\delta) + 8.\text{VC-dim}(H).\log_2(13/\varepsilon))$$

Substituting values, $\varepsilon = 0.05$, $\delta = 0.01$, VC-dim(H) = 8

$m >= 1/0.05\ (4.\log_2(2/0.01) + 8.(8).\log_2(13/0.05))$
$m >= 20\ (4.\log_2 200 + 64.\log_2 260)$
$m >= 20(4*7.643856 + 64*8.022368) = 10880.1395$
$m >= 10881$

Also a lower bound on the sample complexity is given by the following equation where if m is lesser than the quantity, then L will output a hypothesis with error greater than 5% with probability greater than 1%.

$$m < \max [(1/\varepsilon) \log(1/\delta), (\text{VC-dim}(C) - 1)/32\varepsilon]$$

Substituting values,

$m < \max [(1/0.05) \log(1/0.01), (8 - 1)/32(0.05)]$

$m < \max [20 \log 100, 140/32]$

$m < \max [40, 4.375]$

$m < 40$

where ε is the upper bound on the true error of the hypothesis over the distribution of instances and δ is the upper bound on the probability of the learner failing to learn an accurate hypothesis (one with error within bounds).

The sample complexity m, grows logarithmic & linear in ε and only logarithmic in δ**.**

The second equation says that L needs at least 40 training samples to prevent the learner from outputting a hypothesis with error > ε with probability greater than δ.

The sample complexity m tells that the learner requires at least 10881 training samples to PAC learn a model that with probability at least (1-δ) will output a hypothesis h ∈ H such that error over the entire distribution of instances (i.e true error) is less than or equal to ε.

3. Suppose you are given X, which is the set of binary strings of length 5. Let C be the set of classifiers over X where each classifier consists of 1, 0, or * (* matches both 0 and 1) e.g. c = * * * * 0 returns true for any string ending in 0, and false for all others. Similarly, c = * * * * * returns true for all strings. Does C shatter the set X? Support your answer by proof.

<u>ANSWER:</u>

A set of instances D is shattered by a hypothesis space H iff for every dichotomy of D there is a hypothesis in H consistent with this dichotomy.

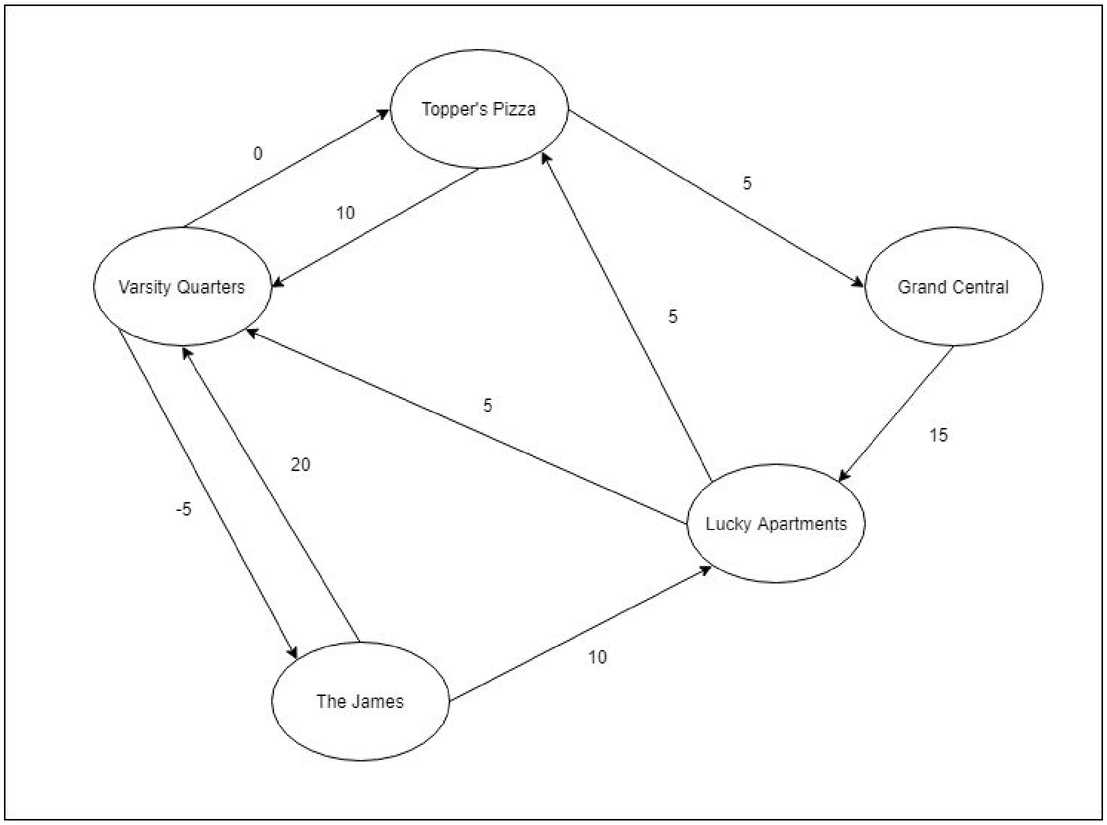
According to the problem definition, any classifier in C classifies strings based on matching of bits at specific position (as given by the classifier definition).

C does not shatter X. Let us consider the following training samples.

String	**Class / Label**
01010	+
10101	+
…… (All other strings)	-

There is no hypothesis in C that is consistent with the above set of training samples. I.e there is no single classifier that uniquely could match only the strings “01010” and “10101”. The only classifier “*****” matches both “01010” and “10101” and hence could predict them as positive, but on the hind-side the classifier also would match any other string under consideration (length-5 binary string) and predict them as positive, no correct negative predictions will be made.

<u>Part 2 - Reinforcement Learning</u>



Suppose you are a delivery driver for Topper's Pizza, and that your delivery route is restricted due to construction to the graph depicted in the diagram. The edges between locations represent tips (rewards) for each delivery. Take this as your deterministic reinforcement environment with γ=0.1, and further assume we are learning a Q-table where all initial values are 5.

1. Assume you follow the path Topper's Pizza --> Grand Central --> Lucky Apartments --> Varsity Quarters --> Topper's Pizza. Using one-step, standard Q learning, show the calculations that produce the new Q-table entries for each state-action pair (e.g. Topper's Pizza --> Grand Central) on this path.

ANSWER:

Q-table initialized with default values.

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	5	5	5	5
Grand Central	5	5	5	5	5
Lucky Apartments	5	5	5	5	5
Varsity Quarters	5	5	5	5	5
The James	5	5	5	5	5

After the state transition Topper’s Pizza → Grand Central

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Topper’s Pizza, a = Move to Grand Central, r = 5, s’ = Grand Central

Q’(Topper’s Pizza, Grand Central) = 5 + 0.1 * (5) = 5.5

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	**5.5**	5	5	5
Grand Central	5	5	5	5	5
Lucky Apartments	5	5	5	5	5
Varsity Quarters	5	5	5	5	5
The James	5	5	5	5	5

After the state transition Grand Central → Lucky Apartments

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Grand Central, a = Move to Lucky Apartments, r = 15, s’ = Lucky Apartments

Q’(Grand Central, Lucky Apartments) = 15 + 0.1 * (5) = 15.5

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	**5.5**	5	5	5
Grand Central	5	5	**15.5**	5	5
Lucky Apartments	5	5	5	5	5
Varsity Quarters	5	5	5	5	5
The James	5	5	5	5	5

After the state transition Lucky Apartments → Varsity Quarters

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Lucky Apartments, a = Move to Varsity Quarters, r = 5, s’ = Varsity Quarters

Q’(Lucky Apartments, Varsity Quarters) = 5 + 0.1 * (5) = 5.5

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	**5.5**	5	5	5
Grand Central	5	5	**15.5**	5	5
Lucky Apartments	5	5	5	**5.5**	5
Varsity Quarters	5	5	5	5	5
The James	5	5	5	5	5

After the state transition Varsity Quarters → Topper’s Pizza

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Varsity Quarters, a = Move to Topper’s Pizza, r = 0, s’ = Topper’s Pizza
Q’(Varsity Quarters, Topper’s Pizza) = 0 + 0.1 * (5.5) = 0.55

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	**5.5**	5	5	5
Grand Central	5	5	**15.5**	5	5
Lucky Apartments	5	5	5	**5.5**	5
Varsity Quarters	**0.55**	5	5	5	5
The James	5	5	5	5	5

The above is the final Q-table after traversing the path.

2. After your learning from the previous path, what will be the next two apartments, in order, that you will visit if you perform no exploration steps? Again, show the results of one-step, standard Q learning from taking this path (calculations of Q values for each state-action pair).

ANSWER:

Given that we are at Topper’s Pizza at the end of the traversal, the next candidate action (using exploitation strategy) would be **moving to Grand Central**.

After the state transition Topper’s Pizza → Grand Central

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Topper’s Pizza, a = Move to Grand Central, r = 5, s’ = Grand Central
Q’(Topper’s Pizza, Grand Central) = 5 + 0.1 * (15.5) = 6.55

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	**6.55**	5	5	5
Grand Central	5	5	15.5	5	5
Lucky	5	5	5	5.5	5

Apartments					
Varsity Quarters	0.55	5	5	5	5
The James	5	5	5	5	5

From Grand Central, the next profitable move is to Lucky Apartments.

Q’(s,a)← r + γ * max across all a’ Q’(s’ ,a’)

S = Grand Central, a = Move to Lucky Apartments, r = 15, s’ = Lucky Apartments
Q’(Grand Central, Lucky Apartments) = 15 + 0.1 * (5.5) = 15.55

State\Action	Topper’s Pizza	Grand Central	Lucky Apartments	Varsity Quarters	The James
Topper’s Pizza	5	6.55	5	5	5
Grand Central	5	5	**15.55**	5	5
Lucky Apartments	5	5	5	5.5	5
Varsity Quarters	0.55	5	5	5	5
The James	5	5	5	5	5

3. Let c = 10. Given that you start at Topper's Pizza, perform an exploration step by choosing the next apartment probabilistically (i.e. calculate the probabilities for all possible actions). In the case of a tie, choose the apartment alphabetically. Show your work.

ANSWER:

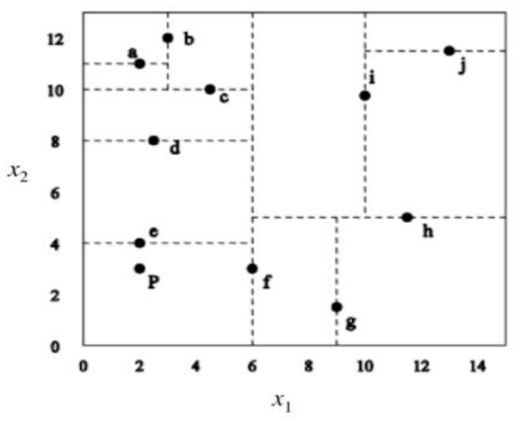
s = Topper’s Pizza, c = 10
P(a_i | s) = c ^ Q’(s, a_i) / sum of c ^ Q’(s, a_j) for all j (existing state-action transition pairs from s)
The value of the denominator in the above equation = 10^5 + 10^6.55 = 3648133.89

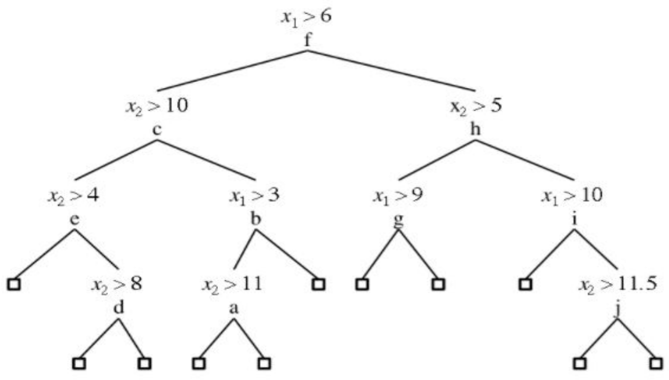
P(Grand Central | Topper’s Pizza) = 10^6.55 / 3648133.89 = **0.972588726**
P(Varsity Quarters | Topper’s Pizza) = 10^5 / 3648133.89 = 0.0274112746

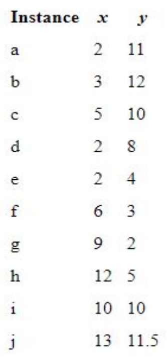
Probabilistically, the next apartment to be visited is **Grand Central.**

Part 3 - Nearest Neighbors with a k-d tree

Using the k-d tree and the training set displayed in the file kd-tree.png, show how the nearest neighbor for x(q) = (7, 10) would be found. Assume that, for all instances in the figure, the features (i.e. coordinates) have integer values. For each step in the search, show the distance to the current node, the best distance found so far, the best node found so far, and the contents of the priority queue. You should use Euclidean distance and assume that the coordinates of the instances in the plot are those shown in the table given in the figure.







Instance	x	y
a	2	11
b	3	12
c	5	10
d	2	8
e	2	4
f	6	3
g	9	2
h	12	5
i	10	10
j	13	11.5

$x^{(q)} = (7, 10)$

Action	distance	best distance	best node	priority queue
		+∞		(f, 0)
Pop f	√50	√50	f	(h, 0) (c, 1)
Pop h	√50	√50	f	(i, 0) (c, 1) (g, 5)
Pop i	√9	√9	i	(c, 1) (j, 3) (g, 5)
Pop c	√4	√4	c	(e, 0) (b, 0) (j, 3) (g, 5)
Pop e	√61	√4	c	(b, 0) (d, 0) (j, 3) (g, 5)

Pop b	√20	√4	c	(d, 0) (j, 3) (a, 4) (g, 5)
Pop d	√29	√4	**c**	(j, 3) (a, 4) (g, 5)
Pop j				(g, 5)
Return c				

Nearest neighbour = c

<u>Part 4 - Markov Chains</u>

1. Why are sampling algorithms of the Markov Chain Monte Carlo (MCMC) family particularly useful? Specifically, what do they allow us to accomplish?

<u>ANSWER:</u>

The sampling algorithms of the Markov Chain Monte Carlo (MCMC) family are useful to sample from a higher dimensional complex probability distribution. In case where the prior distribution and likelihood are described by proper probability distribution (like bell curves), we can model the **posterior probability distribution of the data**. If the curves are not convenient, we cannot compute them analytically and hence resort to MCMC approximation methods. MCMC methods help us to estimate the shape of the posterior distribution in case we can't compute it directly. For this reason, MCMC family of algorithms are particularly useful.

Markov Chain Monte Carlo methods are used to approximate the **posterior distribution of a parameter of interest by random sampling in a probabilistic space**. Monte Carlo simulations provide a way of estimating a fixed parameter by generating random numbers repeatedly. By taking the random numbers generated and doing some computation on them, Monte Carlo simulations provide an approximation of a parameter where calculating it directly is **impossible or extremely expensive**. Monte Carlo simulations aren't only used for estimating the area of difficult shapes. By generating a lot of random numbers, they can be used to model very complicated processes. In practice, they're used to forecast the weather, or estimate the probability of winning an election.

2. How does the number of iterations (samples) relate to your answer from above?

In order to sample the true distribution, we need to be able to convert (or model) the target distribution as a **stationary distribution** using Markov chains. For a Markov chain to be able to reach a stationary distribution (where the subsequent actions/events do not change the state transition probabilities), **sufficient number of iterations of random walk over the graph** of samples must have happened. If the Markov chain picks a random instance and updates the state transition probability, it might be biased based on the starting point. In order **to overcome the bias introduced by the starting point**, we perform as many iterations of picking random samples such that there are a lot of starting points, one for each iteration such that no subset of starting points affect the probability distribution. Since an exact lower bound is difficult to come up with, we say that a sufficiently large number of iterations need to be performed until the probability distribution converges to a stationary distribution.

3. Why is ergodicity an important property of Markov Chains for MCMC algorithms? Again, comment on how it relates to your answer from above.

For a Markov chain to be ergodic, it needs to be irreducible and aperiodic.
Irreducible chain: can get from any state to any other eventually(non-zero probability)
Periodic state: state i is periodic with period k if all returns to i must occur in multiples of k
Ergodic chain: irreducible and has an aperiodic state. Implies all states are aperiodic, so chain is aperiodic.

The Markov chain being irreducible gives the chance for the **random walks to visit all the instances** (nodes) in the graph. Also, the chain being aperiodic ensures that the **random walks can explore all possible states** in the chain and are not localized or caught up in a subset of states thus allowing the possibility to reach the expected stationary distribution in a very large number of iterations. Thus being ergodic (irreducible and aperiodic) is an important property of Markov chains for MCMC algorithms so that they are able to reach the true distribution at least approximately at the end of large number of samples and performing random walks from each of those samples along the instance graph and updating the state transition probability matrix.