CIS 625 – Homework 2

Prasanna Poudyal	Jiahang Sha	Abdullah Zaini	Lingqi Zhang
poudyal	jhsha	azaini	lingqiz

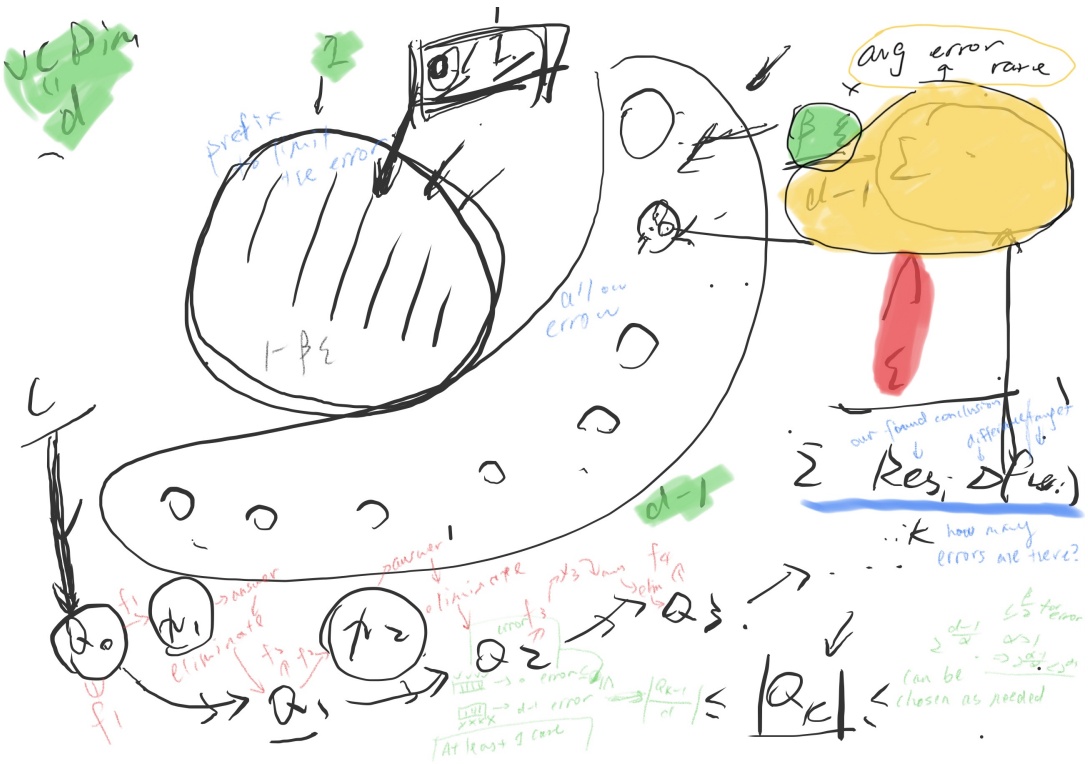
November 24, 2020

This assignment was typed up on an overleaf document, so multiple people were able to edit the same question. We thus clarified who the primary writer of each solution was and who the secondary writers/reviewers were at the beginning of each question.

Problem 1

Primary typer: J.S., Secondary typers/reviewers:

Let $\epsilon > 0$. Assume that the points can be shattered into $S_1, ..., S_d$ with VC dimension d. Set $DIST$ to be a distribution that for $S_1, ..., S_d$ each of them has weight $\frac{\beta\epsilon}{d-1}$ and S_n has weight $1 - \beta\epsilon$. We limit the large part $f(S_n)$ to be prefixed and then choose a subset of concepts Q_0 with $|Q_0| = 2^{d-1}$ that are consistent with the prefixed $f(S_n)$, so that in the future queries the algorithm only needs to conclude within this subset of concepts some result as RES_i corresponding to each S_i that satisfies $\beta\epsilon\frac{\Sigma_{i=1}^{d-1} Res_i \Delta f(S_i)}{d-1} \leq \epsilon$ with $f(S_i)$ being the target function at S_i. From Q_0, the algorithm performs elimination process based on the response from each statistical query, meaning Q_1 should have the only have the concepts in Q_0 that are consistent with the first query, Q_2 should have the only have the concepts in Q_1 that are consistent with the second query, and so on so forth. With facilitation of this elimination process, the algorithm will be able to compare the value of target concept newly chosen from the current subset of elimination to compare against the response from queries. Notice that when the algorithm makes a query, the best situation is that the chosen target concept is completely the same as the response from the query at all shattered sets, and the worst situation is that the chosen target concept is different from the query at all shattered sets, which there are $d - 1$ of them excluding the fixed one. By the number of differed values between the chosen target concept and the query response, we observe that the one query can divide a current subset of elimination into $d - 1 - 0 + 1 = d$ subsets. Among these, the new subset of elimination must be taking at least 1 case, and therefore it is possible that $|Q_i| \geq |Q_{i-1}|/d \geq \frac{|Q_0|}{d^i} = \frac{2^{d-1}}{d^i}$. Because we need another upper bound on the size of this subset to get a lower bound on the times of elimination i (and therefore has made i statistical queries), we assume $|Q_i|$ is upper bounded by $2^{\frac{d-1}{\alpha}}$ for some $1 < \alpha \leq \frac{\beta}{2}$. This is possible because if this is not true then we can just continue the elimination process until it gets below it. Then, $2^{\frac{d-1}{\alpha}} \geq \frac{2^{d-1}}{d^i}$ gets us $i \geq (\frac{\alpha-1\log 2}{\alpha}\frac{d-1}{logd}) = \alpha'\frac{d}{logd}$ for some constant α', meaning that the algorithm needs to make SQ for $\alpha'\frac{d}{logd}$ times. Suppose for contradiction that the algorithm does not make as many SQ's as $\alpha'\frac{d}{logd}$. Then, $|Q_i| \geq 2^{\frac{d-1}{\alpha}}$, from which we choose our the current target concept f_{i+1}. Notice that this is equivalent as seeing the size of a concept class with $\frac{d-1}{\alpha}$ variables each taking 0 or 1. Then, the expected value in this alignment error can be approximated through $\frac{\Sigma_{j=1}^{d-1} Res_j \Delta f_i+1(x_j)}{d-1} \geq \frac{\Sigma_{i=1}^{\frac{d-1}{\alpha}} C_i^{\frac{d-1}{\alpha}} \cdot i}{\Sigma_{i=1}^{\frac{d-1}{\alpha}} C_i^{\frac{d-1}{\alpha}}} = \frac{\frac{d-1}{\alpha} 2^{\frac{d-1}{\alpha}-1}}{2^{\frac{d-1}{\alpha}}} = \frac{d-1}{2\alpha} \geq \frac{d-1}{\beta}$ with the choice of alpha, meaning that it is possible for $\beta\epsilon\frac{\Sigma_{i=1}^{d-1} Res_i \Delta f(x_i)}{d-1} > \epsilon$ to occur, which is undesirable.



The picture above illustrates the general idea how the elimination process works to create desired conditions. The relation of the "bigger" set and "smaller" sets by the constructed distribution are the circles in the middle, and to their right is illustration of how the error can be upper bounded using this construction. The flow line on the bottom illustrates how the set of concepts are shrunk. More annotations are available with zoom in with colors indicating the relation between the values.

Problem 2

Primary typer: Lingqi , Secondary typers/reviewers: Prasanna

We aim to iteratively build a decision list that is consistent with the observed data. Assume the input x are binary strings of length n. For convenience, we will use $x_i,\ i \leq n$ to refer to the i^{th} bit, and $x_i,\ n+1 \leq i \leq 2n$ to refer to the negation of the $(i-n)^{th}$ bit.

We will represent decision list as two array of length at most $2n$:
cond which represents the sequence of tests, and
out which represents the corresponding Boolean output.

Our algorithm is the "SQ variant" of the consistency algorithm we have discussed in class. To test for consistency at each "node", we ask for (τ is the tolerance parameter in SQ):
$Pr[(x_i = 1)\&(y = b)] \geq 1 - \tau$ **and** $Pr[(x_i = 1)\&(y = \neg b)] \leq \tau$.

In addition, we "filter" samples by adding the additional clause to our SQ that everything already in the decision list has to be set to 0:
$x_j = 0,\ \forall x_j$ in **cond**

The full detail of the algorithm is as follows:

Algorithm 1 SQ Learning Decision List

```
1:  procedure L(ε, τ, SQ)
2:      cond ← [ ]                                   ▷ Initialize with empty list
3:      out ← [ ]
4:      for l from 1 to 2n do                        ▷ Build a decision list of length at most 2n
5:          for i from 1 to 2n do                    ▷ We will add one item to the list at a time
6:              if x_i not in cond then              ▷ Test does not repeat in decision list
7:                  p_1 ← SQ[(x_j = 0 ∀x_j in cond) & (x_i = 1) & (y = 1)]
8:                  p_2 ← SQ[(x_j = 0 ∀x_j in cond) & (x_i = 1) & (y = 0)]
9:                  if (p_1 ≥ 1 − τ) & (p_2 ≤ τ) then   ▷ Condition for local consistency
10:                     cond append x_i              ▷ Add test to decision list
11:                     out append 1                 ▷ Add output to decision list
12:                     continue                     ▷ Jump to next iteration of the outer loop
13:                 end
14:                 if (p_1 ≤ τ) & (p_2 ≥ 1 − τ) then
15:                     cond append x_i
16:                     out append 0
17:                     continue
18:                 end
19:             end
20:         end
21:     end
22:     return {cond, out}
```

The maximum number of statistical queries (SQ) required in our algorithm is less then $2n * 2n * 2 = 8n^2$. Thus, with a polynomial number of SQ, our algorithm is able to find a hypothesis decision list that is **consistent** with observed data. The rest of the analysis is the same as the consistency principle we discussed in PAC learning.

Problem 3

Primary typer: Prasanna , Secondary typers/reviewers: Lingqi, J.S.

I. Claim: Modified SQ learning $\Rightarrow$ CN learning
Justification: We note that the proof given in class for SQ learning $\Rightarrow$ CN learning still works here. In particular, to simulate SQ with CN, We first define $X_1 = \{x \in X : \chi(x,0) \neq \chi(x,1)\}$ and $X_2 = X - X_1$. We need the following probabilities:

- P_1 over X_1: $P_1[x] = P[X]/P[X_1]$
- P_2 over X_2: $P_2[x] = P[X]/P[X_2]$
- P^1_χ: $P_1[\chi(x,y) = 1]$
- P^2_χ: $P_2[\chi(x,y) = 1]$
- $\tilde{P}_\chi$: $P[\chi(x,\tilde{y}) = 1]$

Since we now have direct access to unlabeled instances x, both P_1, P_2, and P^2_χ can be directly estimated by drawing a set of unlabeled x samples. To simulate SQ:

- We compute $P_\chi = \frac{\tilde{P}_\chi}{1-2\eta} + (1 - \frac{1}{1-2\eta}) P_2 P^2_\chi - \frac{\eta}{1-2\eta} P_1$ for any statistical queries (the sensitivity analysis stays the same).
- We can return unlabeled instances of x directly if the algorithm asks for them, since x is not corrupted by noise in the CN model.

II. Claim: Rectangles in the real plane are learnable in modified SQ model (and thus in the CN model). WLOG, we assume samples within the rectangle are positive.
Algorithm:

Algorithm 2 Learning algorithm for axis-aligned rectangles in real plane

procedure $\mathcal{L}(\epsilon, \delta, \tau, \text{SQ})$
  $X \leftarrow$ *unlabeled instances* ▷ Draw a set of unlabeled x
  $h \leftarrow$ `false` ▷ Initial hypothesis that's always false
  for `each x` $= (x_1, x_2) \in X$ **do** ▷ Get a positive example
    `rect` $\leftarrow$ `rect that only contains x`
    $p \leftarrow$ `SQ` $(x' \in$ `rect` $\Rightarrow y = 1)$
    if $p \geq 1 - \tau$ **then**
      $x_{pos} \leftarrow x$
      $h \leftarrow$ `rect`
      `break for`
    end
  end
  if x_{pos} `is empty` **then**
    return h
  end
  `direction` $\leftarrow$ `North` ▷ Current direction: start with maximum y coordinate (north)
  while all points are not considered **do**
    `rect' ← Expand rect in current direction to include the closest point to` x_{pos}
    $p \leftarrow$ `SQ` $(x' \in$ `rect` $\Rightarrow y = 1)$
    if $p \geq 1 - \tau$ **then**
      `rect` $\leftarrow$ `rect'`
    else
      `Change direction clockwise. North -> East, East -> South, etc.`
    end
  end
  $h \leftarrow$ `rect`
  return h

Above, $\texttt{SQ}(\chi(x, y))$ is the oracle that returns the probability that $\chi(x, y)$ holds.

Proof of correctness: The first loop essentially identifies one positive example from the collection of unlabelled X instances. We instantiate a rectangle small enough that it only contains only that point. We know this is possible in the real plane since a single point is trivially a rectangle. Next, we query this rectangle and ask if x belongs to this rectangle implies y = 1. Note that if and only if the SQ oracle returns a probability value $\geq 1-\tau$ then we know that point x in this rectangle is a positive example because otherwise, the return value would have to be $\leq \tau$.

After repeating the above process for all m points in the unlabeled sample if we don't get any positive example, we know w.h.p that the target rectangle is either an empty set or contains probability weight $\leq \epsilon$ so returning a hypothesis that always predicts false has zero training error and has true error $\leq \epsilon$.

Next, we expand this zero measure rectangle in each of the four directions, one direction at a time. We start by expanding the rectangle to the closest point north of x_{pos} and querying again. If the probability returned by SQ oracle still lies within $1 - \tau$ to 1 inclusive, we know that this new point is also a positive example so we expand our hypothesis rectangle to includ this new point and repeat for the next closes point up north. Otherwise, we establish that x_{pos} is the northern most point, i.e., it has the max y coordinate among all positive examples in the sample and then we consider next direction.

We note that at any point when we expand the rectangle and query it, if we obtain $1 - \tau$ from SQ, we know that the rectangle is a subset of the target rectangle.

Finally, we use "kernel trick" to learn axis non-aligned rectangles. The algorithm will first use recursive multisection search to determine a good angle to rotate the data points before the learning algorithm starts. Letting $\Delta > 0$ such that $\frac{\pi}{\Delta}$ is bounded within polynomial of n, we partition the set $\{0, \pi\}$ into $\{\{0, \Delta\}, \{\Delta, 2\Delta\}, \ldots \{\pi - \Delta, \pi\}, \}$ and pick a θ from each of these set to rotate the set of unlabelled X by an angle of θ before running the algorithm. This will be valid because the algorithm is in search of the smallest target rectangle that fit the data. When a rotation is successful, we notice that with the rotation by an angle at some interval, the number of correctly rotated data that the target rectangle can precisely describe will be no less than the number of less correctly rotated data that the corresponding target rectangle can describe. When θ is picked from a set $\{i\Delta, (i+1)\Delta\}$, we can further partition this set in a similar strategy until there exist at least three (or more if a precise angle is desired) angles by which rotating the data will not incur target rectangles that label the data differently. Notice that when rotating by at most π, there will be twice when the data can possibly aligned with the axis in the first rotation. In this scenarios where two intervals that differ around by $\pi/2$ both give an ideal rotation, we can do this search on both of them (and even the neighbors of these two as polynomial condition will not be violated) just in case the randomly picked θ is not as near to the optimal rotation angle as being perfectly in the same subset partitioned by Δ. Using this strategy allows the algorithm to preprocess and align (or at least slightly skewed with no negative impact on the target rectangle) the data against the axes.

Runtime: Assuming each query takes polynomial time, we note that sampling m unlabeled instances takes time polynomial in m. Next, to find the first positive example from the initial round of queries (first for loop) also takes polynomial time. Furthermore, expanding the candidate rectangle and querying still takes time polynomial in m. Finally, guessing θ for axis non-aligned target rectangles also takes polynomil time given that Δ is a polynomial function of m, δ and ϵ.

Problem 4

Primary typer: Abdullah, Secondary typers/reviewers: J.S.

Definitions:

Let c_1, c_2 be two different concepts in C such that $c_1,\ c_2 \in C,\ \ c_1 \neq c_2$. Also let L be an algorithm that PAC-learns C. Note. For this example, we could simplify by looking at a case where C only contains concepts c_1 and c_2 and find a contradiction with that class however this is not explicitly needed.

Let a, b, c, d be 4 regions across the sample space such that:
$a = c_1 \wedge c_2$
$b = c_1 \wedge \neg c_2$
$c = \neg c_1 \wedge c_2$
$d = \neg c_1 \wedge \neg c_2$
Let δ be an arbitrary positive quantity, or more formally, $\delta > 0$
Let $\epsilon' = \epsilon + \delta$

Proof:

We will now prove that allowing $\eta > \frac{\epsilon}{1+\epsilon}$ will allow for the construction of a distribution P over a concept c_1 from C such that the malicious PAC model is able to reliably trick the learning algorithm into achieving an error of larger than ϵ. We begin by setting $\eta = \frac{\epsilon}{1+\epsilon} + \delta$. Next we define a distribution P over c_1 such that P gives 0 weight to region c and weight of ϵ' to region b. The rest of the weight of P is distributed evenly among regions a and d. We then instruct the malicious PAC model to provide nothing but negative samples that fall in b (resembling how b is represented according to c_2) as the input to the learning algorithm which occurs with probability η.

We now note that the total number of positive samples in b (which resemble how c_1 labels points in b) $= (1 - \eta) \cdot \epsilon' = (\frac{1}{1+\epsilon} - \delta) \cdot \epsilon' = \frac{\epsilon'}{1+\epsilon} - \delta \cdot \epsilon'$

It then follows that the total number of negative samples in b (which resemble how c_2 labels points in b) $= \eta = \frac{\epsilon}{1+\epsilon} + \delta$

Thus, the total number of negative samples in b outweighs the total number of positive samples in b (proof below).

weight of positive samples < weight of negative samples
$\frac{\epsilon'}{1+\epsilon} - \delta \cdot \epsilon' < \frac{\epsilon}{1+\epsilon} + \delta$
$\frac{\epsilon}{1+\epsilon} + \frac{\delta}{1+\epsilon} - \delta \cdot \epsilon' < \frac{\epsilon}{1+\epsilon} + \delta$
$\frac{\epsilon}{1+\epsilon} + \frac{\delta}{1+\epsilon} < \frac{\epsilon}{1+\epsilon} + \delta$ (since $\delta > 0,\ \epsilon' > 0$)
$\frac{\delta}{1+\epsilon} < \delta$ (since $\epsilon > 0$)

Using the fact that the total weight of negative samples in b is larger than the total weight of positive samples in b which leads us to 1 of 2 cases.

Case 1: L learns c_2. In this scenario, the true error achieved is the entirety of b which has true weight ϵ' which we defined such that $\epsilon' > \epsilon$. Thus this scenario makes the algorithm achieve an error larger than ϵ.

Case 2: L learns c_1. An unlikely case since there are more examples in b that resemble c_2 than c_1, but in the event that this occurs we can show that if L attempts to learn c_2 in the same setting then L will learn c_1 instead. Thus the true error achieved in that scenario is also the entirety of b which has true weight ϵ' which we defined such that $\epsilon' > \epsilon$. Thus this scenario makes the algorithm achieve an error larger than ϵ.

Therefore allowing $\eta > \frac{\epsilon}{1+\epsilon}$ will allow for the construction of a distribution P over a concept c_1 from C such that the malicious PAC model is able to reliably trick the learning algorithm into achieving an error of larger than ϵ. Thus the relationship $\eta \leq \frac{\epsilon}{1+\epsilon}$ must hold in order to achieve error ϵ in the malicious PAC model.

Problem 5

Primary typer: Abdullah, Secondary typers/reviewers:

Definitions:

Let L be an algorithm that PAC-learns $\mathcal{C}$.

Let x represent the polynomial p($1/\epsilon$, n) that bounds the running time and sample size of L to PAC-learn $\mathcal{C}$.

Proof:

We will now show that there is a constant $\alpha > 0$ such that $\mathcal{C}$ is learnable in the malicious PAC model provided that $\eta \leq \alpha/x$ by using L to learn this model.

In the malicious PAC model, a sample $< \tilde{x}, \tilde{y} >$ is drawn i.i.d. from a distribution P over C where $\tilde{y} = c(\tilde{x}) >$ with probability $1 - \eta$. With probability η on the other hand, the samples $< \tilde{x}, \tilde{y} >$ are not chosen in an i.i.d. manner. Our algorithm L takes as input a sample S of size x but it also requires the x,y pairs in the sample to be generated i.i.d from a distribution P over C for it to return a model with error of at most ϵ. Thus all of the samples that are fed to L must meet this criteria. The probability of all x samples being generated in this manner is $(1 - \eta)^x$. Capping η such that $\eta \leq \frac{1}{x}$ will make the probability of getting a good sample, a sample where all x samples are generated i.i.d from a distribution P over C, approach $\frac{1}{e}$ as x gets larger (since $(1 - \frac{1}{x})^x = \frac{1}{e} \simeq 0.368$ for large enough x). This means that L will be fed a good sample, and return a PAC model, with probability 0.368 as the sample size gets larger, which isn't that likely but is still possible.

It also follows that if we are guaranteed that $\eta \leq \frac{\alpha}{x}$ for a constant $\alpha > 0$, setting α to something very small like $\alpha = \frac{1}{200000} = 0.000005$ will make the probability of a good sample being supplied $= (1 - \frac{1}{200000 \cdot x})^x$. We know that x must be at least 1 since we cannot supply a non positive number of samples and then expect the learning algorithm to learn something. Thus the probability of a good sample will begin as $1 - \frac{1}{200000} = 0.999995$ and it will increase as the sample size increases. With this α, the probability of getting a good sample approaches $(\frac{1}{e})^{200000} \simeq 0.999995$ as x get sufficiently large. In this scenario, making α represent a very small constant will increase the probability of getting a good sample to being very very likely.

Thus we have shown scenarios where our PAC-learnable algorithm C is still learnable in the malicious PAC model with some non-trivial probability when η adheres to a certain upperbound that's inversely proportional to the sample size bound. We have also shown how adjusting a constant α can turn the probability of getting a good sample into a very likely probability. Therefore we have shown that that there is a constant $\alpha > 0$ such that a PAC-learnable class $\mathcal{C}$ is learnable in the malicious PAC model provided that $\eta \leq \alpha/x$.

Problem 6

Primary typer: Prasanna, Secondary typers/reviewers: Lingqi

Algorithm 3 Learning algorithm for monotone DNF with membership queries

```
1: procedure L(ϵ, δ, S)
2:     h ← 0                                  ▷ Initial hypothesis that always evaluates to false
3:     for positive example x do
4:         if h(x) = 0 then
5:             for each x_i ∈ x where x_i = 1 do
6:                 x' ← x but with x_i set to 0
7:                 if MQ(x') = 1 then
8:                     x ← x'
9:                 end
10:            end
11:            t ← conjunction of all remaining x_i ∈ x where x_i = 1
12:            h ← h ∨ t
13:        end
14:    end
15:    return h
```

Proof of Correctness:
Claim: The algorithm above produces hypothesis h s.t. the training error of h is 0. We note that the algorithm above essentially for each positive example (x, 1), turns off each index of x, x_i where x_i is turned on in x and tests for membership. If the MQ oracle returns true, we know that x_i doesn't not essential for membership of this example to the target function. Thus, its safe to turn x_i off. FInally, after this process is repeated for every index of x, we are left with only the essential indices, i.e. turning off any of the indices left would render the MQ query false. So, the conjuction of these essential indices is added to the candidate hypothesis.
We note finally that after we are done with all the examples, we obtain a hypothesis that is consistent with the sample because since the target class is monotone, turning off indices can only reduce members. In other words, our process doesn't create any false positives from the sample and the process also ensures that there are no false negatives.
Finally, since we know that this algorithm gives us a consistent hypothesis, it can be used to PAC learn monotone DNF.
Runtime: Each positive example takes at most nt where t is the time taken by MQ for each query. Assuming t is something reasonable (for instance polynomial(, we see that the runtime is of the order mnt, where m is the sample size.
Extensions: This result doesn't imply that non-monotone DNF is learnable in the PAC model with membership queries. This is because the reduction from DNF to monotone DNF could lead us to indeterminate state when we using membership queries and it would take more than polynomial time (at least factorial in n, number of bits) queries to establish membership. More specifically, when using the reduction from DNF to monotone DNF, a bit x_i and $\neg x_i$ reindexed or renamed as x_j could both be turned on simultaneously in the above algorithm in which case the MQ results would not give us any useful information about whether x_i or $\neg x_i$ is essential for membership. We'd have to test each combination of x_i and $\neg x_i$ for all i, so the inner for loop in this fixed algorithm would be really expensive and each for loop could potentially lead to more than one "essential" conjunction.

Problem 7

Primary typer: Lingqi , Secondary typers/reviewers: JS

In the context of classification algorithm, we propose:
I. Any heuristic algorithm using zero-order optimization/search (e.g., simplex algorithm) can be easily simulated with SQ learning.

- The hypothesis is parameterized by $\theta : y = h_\theta(x)$
- The algorithm aims to (iteratively) find a hypothesis h_θ that minimizes $\epsilon(h_\theta)$ on training data. In each iteration i of the optimization, only the value of $\epsilon(h_{\theta_i})$ is required.
- $\epsilon(h_{\theta_i})$ can be estimated with a SQ:
  $\hat{\epsilon}_i = Pr[\chi(x, y) = 1]$, where $\chi(x, y)$ is $h_{\theta_i}(x) \neq y$.
- The optimization routine propose θ_{i+1} based on the history of $\{\theta_{1,2,\ldots,i}\}$ and $\{\hat{\epsilon}_{1,2,\ldots,i}\}$.
- Repeat until convergence.

II. We further propose, if we allow a more general definition of SQ as $E_{(x,y)}[g(x, y)]$, many algorithm using first-order optimization (e.g., gradient descent) can be simulated with SQ.

- WLOG, consider cross-entropy loss for binary classification:
  $J(\theta) = -E_{(x,y)}[y \log f_\theta(x) + (1 - y) \log(1 - f_\theta(x))]$
- In each iteration i, the estimate of the gradient $\frac{dJ}{d\theta_i}$ is:
  $-E_{(x,y)}[y \frac{1}{f_{\theta_i}(x)} \frac{df}{d\theta_i} + (1 - y) \frac{1}{1-f_{\theta_i}} (-\frac{df}{d\theta_i})]$
  which can be estimated with SQ of the form $E_{(x,y)}[g(x, y)]$.

In both cases, if the optimization algorithm is able to converge within polynomial number of iterations, then we will only need a polynomial number of Statistical Queries.

Problem 8

Primary typer: Abdullah, Secondary typers/reviewers: Prasanna, Lingqi, J.S.

Algorithm 4 Learning algorithm for variable noise rate generalization

1: **procedure** $\mathcal{L}(\epsilon,\ \delta,\ \mathcal{L}',\ \mathrm{S})$
2: $< X, \hat{y} > \leftarrow$ `Variable noise rate sample`
3: `S'` $\leftarrow$ `random and uniform shuffling of` $< X, \hat{y} >$
4: $h \leftarrow \mathcal{L}'(S, \frac{\delta}{|S'|}, \epsilon)$
5: **return** h

Above, $\mathcal{L}'$ is the learning algorithm that correctly learns $\mathcal{C}$ in CN model.

Proof:

In the CN model, the probability of any data sample m_i being corrupted is η. We argue that this is the same when the variable noise rate CN model is run on L. Since the data is drawn i.i.d the probability of any arbitrary but particular $< x, y >$ data sample being corrupted, if it is included in S, is $\sum_{i=1}^{m} \frac{1}{m} \cdot \eta_i = \frac{1}{m} \sum_{i=1}^{m} \eta_i = \eta$ since that one arbitrary sample can be anywhere in the sequence of samples and thus can be affected by any of the η_i's in the predetermined classification noise sequence.

It also follows that in the CN model, the probability of the i'th ordered inputted sample m_i being corrupted is also η. To clarify this means that the probability of the first sample inputted to L' is η and the probability of the i'th sample inputted to L' is also η for $1 \leq i \leq m$. We argue that this is the same when the variable noise rate CN model is run on L. Since we uniformly shuffle the samples after the classification noise is applied to them the probability that the i'th sample was affected by classification noise is $\sum_{j=1}^{m} \frac{1}{m} \cdot \eta_j = \frac{1}{m} \sum_{j=1}^{m} \eta_j = \eta$ since there is a $\frac{1}{m}$ chance that each sample m_j is placed as the i'th element (since its uniformly shuffled) and each sample m_j is corrupted with probability η_j.

Therefore, since running a variable noise rate model on $\mathcal{L}$ mimics the CN model in that (1) the probability of any data sample m_i being corrupted is η and (2) the probability of the i'th ordered inputted sample m_i being corrupted is also η, we can see that if algorithm $\mathcal{L}'$ is capable of learning $\mathcal{C}$ in the CN model, $\mathcal{L}$ will be capable of learning $\mathcal{C}$ in the variable noise CN model. Thus, if $\mathcal{C}$ is learnable in the CN model, it is also learnable in the variable noise rate CN model and algorithm L shows how to use an algorithm that learns $\mathcal{C}$ in the CN model to learn $\mathcal{C}$ in the variable noise CN model.