Homework 4 TTIC 31250

Eddie Shim
(worked partially with Hanqi Zhang)
eddieshim@uchicago.edu

05/18/2020

Problem 1

[PAC learning of small OR functions] Give a polynomial-time algorithm that guarantees to find an OR function of at most $O(r \log m)$ variables that is consistent with the training data, where m is the number of training examples (part 1). Describe a variant of your algorithm (also running in polynomial time) that finds an OR function of only $O(r\log(1/\epsilon))$ variables, with error at most $\epsilon/2$ on the training data (part 2).

Part 1: My approach will be to show that giving a polynomial-time algorithm that guarantees to find an OR function is the same as the greedy set-cover problem. The set cover problem is defined as such[1]: given a universe U of size n, and subsets $S_1, ..., S_k \subseteq U$, we want to find a collection C of these subsets whose union contains the entire universe U, where we try to minimize the size of $|C|$. In the greedy approach, we iterate by choosing the set S_i containing the most uncovered points each time, ending when all points of U are covered. It can be shown that in an optimal solution that contains r sets, the greedy algorithm can find a set cover with at most $O(r\log m)$ sets, which is equivalent in our problem to finding $O(r\log m)$ variables consistent with the training data.

Proof that the greedy set-cover algorithm finds a set cover with at most $O(r\log m)$ sets:
Let our universe U contain m points, and let the optimal set cover have size r. In the first iteration of our greedy algorithm, we will choose a set of size at least $\frac{m}{r}$. In the next iteration, we will now have m_1

[1] From Shuchi Chawla's lecture on Advanced Algorithms: http://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture02.pdf?fbclid=IwAR3_tslGLJA_dUuvGOmKM28ELgjgBMDnsrVOwj7EL7aewT93NOjkGffjbSw

uncovered points in U, where:

$$m_1 \leq m - \frac{m}{r} = m(1 - 1/r)$$

We know that at least one of the remaining sets S_i must have at least $m_1/(r-1)$ of such points, since we know that the optimal set size is at max r sets. In the second iteration, our greedy algorithm thus choose m_2 uncovered points, where:

$$m_2 \leq m_1(1 - 1(r-1)) \leq m_1(1 - 1/r) \leq m(1 - 1/r)^2$$

This shows a general pattern where we can calculate m_i as such:

$$m_{i+1} \leq m_i(1 - 1/r) \leq m(1 - 1/r)^{i+1}$$

To show how many iterations of our greedy algorithm we need to cover the entire set, suppose it takes k iterations and by the k^{th} iteration we have less than 1 point remaining uncovered. Thus:

$$m(1 - 1/r)^k < 1$$

$$m(1 - 1/r)^{r(k/r)} < 1$$

$$(1 - 1/r)^{r(k/r)} < 1/m$$

$$e^{-k/r} < 1/m$$

$$k/r > \ln m$$

$$k < r \ln m$$

Thus, we have proved that the greedy set-cover algorithm takes k steps in order to represent our entire universe by a union of size $O(r \log m))$ sets.

To show that the problem can be represented as a greedy problem, define each "point" in our set as each data point in our training set. For each feature x_j, where x_j is defined as the j^{th} feature out of n possible boolean variables, define a set as the union of all positive points which utilize x_j in order to produce a positive label. From here, first form a set of S of x_j variables that create a negative example. Then for every x_j in set S, delete these chosen x_j literals from all positive examples. Next, we iterate by saying for each data point with a positive label y_k, ($y_1 ... y_p$ where $p \leq m$), and for each x_j in y_k, add y_k to the set S_i. Thus, the set S_i represents all positive examples that features x_j appears in. We've now set up the approach such

that we need to find the union of all S_i than spans all positive y_k in training universe U. Thus, by the greed set-cover algorithm proof described above, we have that we can find an OR function of at most $O(r \log m)$ features that is consistent with the training data.

Part 2: Going back to our proof of the greedy set-cover algorithm, we defined our algorithm as stopping in k iterations where at the end of this iteration, we had < 1 point remaining uncovered. We can modify this algorithm to stop earlier, and instead of 0 training error we now tolerate at most $\epsilon/2$ error. Thus we have the following inequality:

$$m(1 - 1/r)^k < \epsilon m/2$$

$$e^{-k/r} < \epsilon/2$$

$$-k/r < \ln \epsilon/2$$

$$k > r \ln 2/\epsilon$$

Thus, by stopping our greedy algorithm early at $k = O(r \log 1/\epsilon)$ steps, we can guarantee that we can achieve less than $\epsilon/2$ error on the training set.

Problem 2

[Uniform distribution learning of DNF formulas] Give an algorithm to learn the class of DNF formulas having at most s terms over the uniform distribution on $\{0,1\}^n$, which has sample size polynomial in n and s, and running time $n^{O(log(s/\epsilon))}$. So, your algorithm matches the SQ-dimension lower bounds.

To approach this problem, I read up on the following reference[2]. As the first step, we first look into γ-relevance of the terms of a DNF formula, where we define γ-relevance as such: for each term t in DNF formula f with a distribution D, t is γ-irrelevant under D for some $0 < \gamma \leq 1$ iff the probability of drawing a vector from D satisfies $t < \gamma$. If not, then term t is γ-relevant under D.

With this definition, we want to prove the following observation: given a DNF formula, represented by s number of terms t as the following $f = t_1 + ...t_s$, with distribution D and any k, $1 \leq k \leq n$, no term t_i with $|t_i| > \lg k$ is $(\gamma = \frac{1}{k})$-relevant under distribution D.

Note that any term t_i with $|t_i| > \lg k$ is satisfied by $2^{n-|t_i|} > 2^{n-\lg k} = \frac{1}{k}2^n$ examples. Because D is

[2]Learning DNF Under Uniform Distribution in Quasi-Polynomial Time, Verbeurgt 1990

uniform, the probability of drawing an example satisfying term t_i is less than $\frac{1}{k}$, therefore all terms t_i are $(\frac{1}{k})$-irrelevant.
Given the proven observation, we can state that all terms with more than lg $\frac{s}{\epsilon}$ attributes are $\frac{epsilon}{s}$-irrelevant. Since there are s total terms in f, the error introduced by ignoring all terms with more than lg $\frac{s}{\epsilon}$ attributes is at most ϵ. Therefore, we can construct our wanted algorithm that creates a hypothesis h consisting of only the terms t_i of f with at most lg $\frac{s}{\epsilon}$ attributes which has error $\leq \epsilon$. Here are the steps for this algorithm which searches for hypothesis h:

1. Construct the set $M_n^{lg(\frac{s}{\epsilon})}$, which is defined as the set of all terms with at most $lg(\frac{s}{\epsilon})$ literals on n variables

2. Find the smallest subset of $M_n^{lg(\frac{s}{\epsilon})}$ which covers at least a fraction (1 - ϵ) of the positive examples in our training data, without covering any negative examples

To show that we can fulfill step 1 in polynomial time, we can use a similar approach we used in problem 1, namely showing that searching for the set $M_n^{lg(\frac{s}{\epsilon})}$ can be represented by the greedy algorithm of the partial set cover problem, which searches for the near-optimal cover in polynomial time. To show that we can fulfill step 2, namely that the algorithm can cover a fraction (1- ϵ) of a sample of polynomial size, we can use an Occam algorithm which draws a sample of polynomial size, then searches for a hypothesis that covers at least a fraction of (1- $\epsilon/4$) of the training data and produces a hypothesis with high probability of covering at least a fraction (1 - ϵ) of the training data.

Partial set cover problem: The partial set cover problem is defined as following: given inputs of $0 < p \leq 1$, positive real costs $c_1, ..., c_y$, finite sets $S_1, ..., S_y$, and a set S where $|S| = t$ such that $\cup_{1 \leq i \leq y} S_i \subseteq S$ and $|\cup_{1 \leq i \leq y} S_i \geq p \times t$, we have the output $J' \subseteq \{1, ..., y\}$ such that $|\cup_{1 \leq i \leq y} S_i \geq p \times t$ and PCcost$(J') = \sum_{j \in J'} c_j$ is minimized. A $p-$cover for S is defined as the collection of sets S_j such that $j \in J'$.

To transform our search of h into the partial set cover problem, let T be the set of terms with at most lg$(\frac{s}{\epsilon})$ attributes. Let $S_1, ..., S_{|T|}$ be the sets of vectors satisfying the terms of T. Because we know there exists a hypothesis h with error less than ϵ, for some indices $J' = \{i_1, ..., i_s\} \subseteq \{1, ..., |M|\}$ the sets $S_{i_1}, ..., S_{i_S}$ that correspond to the terms of h must have $|\cup_{j \in J'} S_j| \geq (1 - \epsilon)|\cup_{1 \leq j \leq |M|} S_j|$. Assign the costs $c_i = 1$ for $1 \leq i \leq |M|$, and let the PCcost be the length of the resulting hypothesis.

The greedy algorithm is similar to the one described in problem 1, where we start with an empty set h, and add terms $t \in M$ which have the largest number of positive examples in S, and none of the negative examples in S. Add t to h, and remove the positive examples from S which satisfy t. Iterate this process until h is a p-cover of S or $|h| > s(2\ln\, t + 5)$.

Occam algorithm The Occam algorithm for learning DNF allows us to draw form a sample space of polynomial size. We take in an input set of terms T containing (1- $\epsilon/4$)-cover of the example space, and output an h with high probability that is an $(1-\epsilon)$-cover of the example space. The following are steps for executing the algorithm:
Let $l = s(2\ln t+5)$. Then draw a sample S of $t = \frac{12}{\epsilon}\ (\ln\, \frac{2}{\delta} + l\, \ln|T|)$ examples from distribution D (calculated using Chernoff bounds). Find our optimal hypothesis by running the greedy partial set cover algorithm, namely $h =$Greedy$(T, S, s, (1-\epsilon/2))$. Finally, we return the disjunction of the terms in h.
We claim that if the greedy algorithm is run on $t = \frac{12}{\epsilon}\ (\ln\, \frac{2}{\delta} + l\, \ln|T|)$ examples from distribution D, then we will return a set of at most $l = s(2\ln t + 5)$ terms from T with probability at least $1-\delta$, which is a $(1-\epsilon)-$cover of the example space.
Proof of the this claim goes as follows. Suppose there exists a set of at most s terms of T which is an $(1-\epsilon/4)$-cover of the example space. Let S be a sample of size t. We want a value of t that is large enough so that with probability $\geq 1-\delta/2$, the $(1-\epsilon/4)$-cover of th example space is an $(1-\epsilon/2)$-cover of the positive sample S of size t. We know that t is bounded by the probability that the cover is not an $(1-\epsilon/2)$-cover of S, so we have $t \leq e^{-\frac{\epsilon}{12}t} \leq \frac{\delta}{2}$. Thus we have:

$$(1) \qquad t \geq \frac{12}{\epsilon}(ln(\frac{2}{\delta}))$$

From the greedy algorithm, we know that the probability that h is an $(1-\epsilon/2)$-cover of S is $\leq e^{-\frac{\epsilon}{8}t}$. Thus the probability that $h \in H$ possible hypothesis with error $\geq \epsilon$ is an $(1-\epsilon/2)$-cover of S is at most $|H|(e^{-\frac{\epsilon}{8}t})$. Bounding this by $\delta/2$, we get:

$$t \geq \frac{8}{\epsilon}(ln(\frac{2}{\delta} + ln(|H|))$$

Since we defined the greedy algorithm to output a hypothesis h with at most $l = s(2\ln t+5)$ terms, we have:

$$(2) \qquad t \geq \frac{8}{\epsilon}(ln(\frac{2}{\delta} + l\ \ln\ (|T|)))$$

In order to satisfy the inequalities (1) and (2), we must have $t = \frac{12}{\epsilon}\ (\ln\, \frac{2}{\delta} + l\, \ln|T|)$ examples.

Thus, now that we've shown that we can use Occam as well as the greedy partial-set cover algorithm, we finally have our algorithm:

1. Create the set $M_n^{\lg \frac{4s}{\epsilon}}$
2. Return the disjunction of terms calculated by $Occam(M_n^{\lg \frac{4s}{\epsilon}}, D, \epsilon, \delta)$

Thus, this algorithm allows us to find the set $M_n^{lg(\frac{s}{\epsilon})}$ in quasi-polynomial run-time, $n^{O(lg(\frac{s}{\epsilon}))}$ and requires sample size polynomial in n and s.

Problem 3

[SQ learning Decision Lists and Trees] Give an algorithm to learn the class of decision lists in the SQ model (and argue correctness for your algorithm). Your algorithm should work for any distribution D. Be clear about what specifically the queries χ are and the tolerances τ.

In order to define a statistical query, we are looking for two parameters that are defined as such: χ, a function such that $X \times \{-1, 1\} \to \{-1, 1\}$, τ, a tolerance parameter where $\tau \geq 0$, and a statistical query oracle SQ(χ, τ), which returns a value in the range $[\mathbb{E}_{x \backsim D}[\chi(x, c(x))] - \tau, \mathbb{E}_{x \backsim D}[\chi(x, c(x))] + \tau]$ where C is defined as a class of boolean functions $c : X \to \{-1, 1\}$. C is SQ-learnable if there exists an algorithm L such that for every $c \in C$, any probability distribution D, and any $\epsilon > 0$, there is a polynomial $p(,,)$ such that L asks the SQ oracle at most $p(1/\epsilon, n, |c|)$ times, and $1/\tau \leq p(1/\epsilon, n, |c|)$, and the queries q can be evaluated in time $p(1/\epsilon, n, |c|)$, and L outputs a hypothesis h such that $err_D(h) \leq \epsilon$.

Looking back to the algorithm described in lecture 1, we use the following rules to learn a decision list:

1. Start with an empty list
2. Find if-then rule consistent with data
3. Put rule at bottom of list so far, and cross off examples covered, repeat until no examples remain

We can convert this into a statistical query learning algorithm. Notice that to find step 2 of our decision list algorithm, we are essentially looking for a feature x_{good} where $p(y = +|x_j = 1) = 1$ (where y can be $-$ and x_j can be 0 without loss of generality). This ensures that x_{good} can be "crossed off" our decision list and

converted into a logical if/then statement that's consistent with our data. Furthermore, notice that features x_{bad} that we do not want (namely the ones we can't cross off as they aren't consistent with the data) have $p(y = +|x_j = 1) < 1 - 1/n$, where n is our data's sample size. This is intuitively because we can have at least one data row that makes the column inconsistent and unable to be crossed off. Thus, we want our SQ model to be able to distinguish between these two x_j, even when noise τ obscures the oracle. We can now define two queries that allow us to grab the conditional probability we want, since Bayes' formula allows us to say $p(y = +|x_j = 1) = p\frac{(y=+\wedge x_j=1)}{p(x_j=1)}$ Thus we define the following two queries:

$$\chi_1(x) = p(y = + \wedge x_j = 1)$$

$$\chi_2(x) = p(x_j = 1)$$

In order to distinguish the good features (x_{good}) from bad features (x_{bad}) we want to choose a τ that can ensure we can distinguish the two queries' results. Notice that in the most deceitful x_{bad} case, without noise have $\chi_1(x) = 1 - 1/n$ and $\chi_2(x) = 1$. With noise, we have $\chi_1(x) = 1 - 1/n \pm \tau$ and $\chi_2(x) = 1 \pm \tau$. Thus the largest difference we can adversely engineer is:

$$(1) \qquad \chi_1(x) - \chi_2(x) = 1/n - 2\tau$$

On the other hand, if we have x_{good}, we know that without noise, we have $\chi_1(x) = \chi_2(x) = q$. With noise, we have $q \pm \tau$, thus the largest difference we can adversely engineer is:

$$(2) \qquad \chi_1(x) - \chi_2(x) = \tau$$

Thus in order to differentiate between (1) and (2), we solve for the inequality:

$$1/n - 2\tau > 2\tau$$

$$1/n > 4\tau$$

$$\frac{1}{4n} > \tau$$

Thus, we can choose $\tau = \frac{1}{8n}$ to ensure that any column can be determined as x_{good} or x_{bad} based on the difference of $\chi_1(x) - \chi_2(x)$. Using the algorithm described in lecture 1, we can iterate this algorithm by looking for a good x_j each time, then restricting the next search by looking for next column on a restricted set, or namely looking for $p(y = +|x'_j = 1|x_j = 1)$. Thus by using the parameters $\text{SQ}(\chi_1, \chi_2, \tau)$, we can ensure we learn decision lists through the SQ-model.