CS 395T Computational Complexity of Machine Learning

Lecture 9: February 17, 2005 — Lecturer: Adam Klivans

Scribe: HyeRyun Kim

9.1 Random Projection for learning intersecions of halfspaces on the ball

9.1.1 Abstraction

Let's consider working on the unit ball. All points are on $\mathbf{S^{n-1}}, \forall x \in \mathbf{S^{n-1}}$, and the norm of the point is 1 by definition ,$||x|| = 1$. Halfspace h is represented by the normal. We can measure the distance from x to hyperplane using dot product $\langle x, h\rangle$. Assume that $\langle x, h\rangle \geq \rho$ for some ρ. Now, we are going to project data on the space to lower dimensional space. In other words, randomly project via $\mathbf{M}$, a $n \times k$ matrix where $k \approx \frac{\log n}{\rho^2}$. First, apply $\mathbf{M}$ to x and h. By the property of Johnson-Lindenstrass lemma mentioned last class, we know that after projection the distance between x and halfspace is approximately same. The normal to the halfspace is also approximately same. Therefore, the following equation is satisfied in lower dimensional space with high probability.

$$\langle \mathbf{M}^t \cdot h, \mathbf{M}^t \cdot x\rangle \geq \frac{\rho}{2}.$$

Similarly,

$$if \quad \langle h, x\rangle \leq -\rho, \quad then \quad \langle \mathbf{M}^t \cdot h, \mathbf{M}^t \cdot x\rangle \leq -\rho.$$

Therefore, $\mathbf{M}^t \cdot h$ will be a halfspace classifying $\mathbf{M}^t \cdot x$ with margin ρ.

9.1.2 Actual Learning Alrogithm

In mistake bound model, we get the all examples and pick a random projection matrix. Then projcet them all down lower dimensional space. We look for a halfspace in lower dimensional space to classify data using brute-force search.

So, How can we classify the data points coming from higher dimensional space? Before the random projection, h classifies the data by looking at inner product. If $\langle h, x\rangle > 0$ then x is classified into positive. otherwise, x is classified into negative. We know that inner product is preserved by Johnson-Lindenstrass lemma. Therefore, we can say the following: fix h such that $||h|| = 1$. $x \in \mathbb{R}$, projection matrix $\mathbf{M}$ ($n \times k$ matrix), for any $x \in \mathcal{X}$, $0 < c < \frac{1}{2}$,

$$\Pr_M[\langle h, x\rangle - 2c \leq \langle \mathbf{M}^t \cdot h,\ \mathbf{M}^t \cdot x\rangle \leq \langle h, x\rangle + 2c] \quad \geq \quad 1 - e^{-c^2 k}.$$

Therefore, projected halfsapce will be a halfsapce in lower dimensional sapce that classify data correctly. Overall learning time depending on margin is $(\frac{1}{\rho})^{\frac{1}{\rho^2}}$.

9.2 Parity function

Conjunction, Decision list, Decison tree and $\vee$ of halfsapces have its own "LOW" degree PTFs that compute each concept. However, parity function can not be computed by low degree PTFs.

9.2.1 Mistake bounded algorithm for learning parity function

The value of parity function is 1 if odd number bits set 1. Otherwise it is 0. The concept of parity function is parity on unknown subject of n variables. For example, a concept is $C = x_1 \oplus x_4 \oplus x_6$, an example is 100000 and label for this example will be 1 in terms of concept C. I represent example and label as (example , label).
Parity function has the following property. For convenience, let $e_1 + e_2$ and $l_1 + l_2$ denote the vector sum mod 2.

Claim: When examples $(e_1, l_1), (e_2, l_2)$ is given, $(e_1 + e_2, l_1 + l_2)$ is an example of same parity function.

Proof: There are three cases for e_1 and e_2.

Case 1: the positions of 1 at e_1 is totally different from the positions of 1 at e_2. When e_1 adds to e_2, the label is the sum of # of bits set to 1 at e_1 and e_2.
$(e_1, 1) + (e_2, 1) = (e_1 + e_2, 1 + 1 = 0)$: odd+odd= even
$(e_1, 1) + (e_2, 0) = (e_1 + e_2, 1 + 0 = 1)$: odd+even = odd
$(e_1, 0) + (e_2, 1) = (e_1 + e_2, 0 + 1 = 1)$: even+odd = odd
$(e_1, 0) + (e_2, 0) = (e_1 + e_2, 0 + 0 = 0)$: even+even = even

Case 2: the positions of 1 at e_1 ovelap partially with the positions of 1 at e_2. overlapped position of 1 become 0 after $e_1 + e_2$. Therefore label is the sum of # of bits set to 1 at e_1 and e_2 except overlapped 1. If the number of overlapped position of 1 is even, it doesn't affect label. Since the number of 1 except overlapped 1 is even(odd) in a example where the number of 1 is even(odd). Even if the number of overlapped position of 1 is odd, it doesn't affect label. Since the number of 1 except overlapped 1 is odd(even) in a example where the number of 1 is even(odd), The label at e_1 and e_2 is flipped after removing overlapped 1. However the result of summation is preserved.
$(e_1, 1) + (e_2, 1) = (e_1 + e_2, 1 + 1 = 0)$: odd+odd= even $\rightarrow$ even + even = even.
$(e_1, 1) + (e_2, 0) = (e_1 + e_2, 1 + 0 = 1)$: odd+even = odd $\rightarrow$ even+ odd = odd.
$(e_1, 0) + (e_2, 1) = (e_1 + e_2, 0 + 1 = 1)$: even+odd = odd $\rightarrow$ odd + even = odd.
$(e_1, 0) + (e_2, 0) = (e_1 + e_2, 0 + 0 = 0)$: even+even = even $\rightarrow$ odd+ odd = even.

Case 3: the positions of 1 at e_1 is totally same with the positions of 1 at e_2. Every 1 becomes 0 after $e_1 + e_2$. Therefore label shoud be 0. Since e_1 and e_2 are exactly same, just consider two cases.
$(e_1, 1) + (e_2, 1) = (e_1 + e_2, 1 + 1 = 0)$
$(e_1, 0) + (e_2, 0) = (e_1 + e_2, 0 + 0 = 0)$ ■

The algorithm of learning parity function starts with to constrct H $(a_1x_1 + a_2x_2 + \cdots + a_nx_n)$ mod 2. The goal of this algorithm is to discover a_i. At any point of this algorithm, There are some set of examples and labels. When we get the new example, if new example is span of previous examples, we get always correct label due to above property of parity function. Therefore, when we get n linear independent examples we can solve for a_i. We can possibly make mistake only when new example is not a span of previous examples. So, everytime I make a mistake, add a new linearly independent vector to current basis. This algorithm makes at most n mistakes before getting n linearly independent vectors and discovering $a_1, a_2, \cdots, a_n$.

This algorithm has mistake bound $O(n)$. If you have a parity of length k, you could acheive mistake bound $O(k \log n)$ and time 2^n.

9.2.2 Learning parity with noise

This is a kind of learning with respect to uniform distribution with noise. Learning parity with noise briefly means that when we are getting examples, each label is going to flip with some consistent probability η. In detail, the scenario is like this; learner has access to an example oracle. When learner push the butten, learner get an outcome. The outcome is a pair of example and label. This example is uniform, independent string over $(0,1)^n$. The label is the concept of evaluated example without noise. However, this label is flipped with probability η in noise model. This happens independently for each example. It is called random classification noise. The goal of learner is to lern output h, a polynomial time computable function such that $Pr[h(x) = c(x)]$ is large. That means that h has high accuracy with respect to the original concept.

9.2.3 The Algorithm for Learning parity with noise

There is an algorithm for learning parity with classification noise on n variables. This algorithm is PAC model. We learn parity function that depends on only the first $O(\log n \cdot \log \log n)$ bits of input in polynomial time. This algorithm runs in time $2^{O(\frac{n}{\log n})}$ and requires $2^{O(\frac{n}{\log n})}$ labeled examples. This is due to the paper from Blum,Kalai and Wasserman.

Observation : Let $(e_1, l_1), (e_2, l_2), \cdots, (e_s, l_s)$ be uniformly distributed examples labeled and corrupted by random noise of rate η. Then $l_1 + \cdots + l_s$ is the correct label of $e_1 + \cdots + e_s$ with probability $\frac{1}{2} + \frac{1}{2}(1-2\eta)^s$.

Proof:
Proof by induction on s. $s = 1$: This is trivial. $\frac{1}{2} + \frac{1}{2}(1-2\eta) = 1 - \eta$.
Now assume that the lemma is true for $s-1$. Given s examples, the probability that $l_1 + \cdots + l_s$ is the correct label of $e_1 + \cdots + e_s$ is $(1-\eta)(\frac{1}{2} + \frac{1}{2}(1-2\eta)^{s-1}) + \eta(\frac{1}{2} - \frac{1}{2}(1-2\eta)^{s-1}) = \frac{1}{2} + \frac{1}{2}(1-2\eta)^s$.
■

The idea of the algorithm is to try to get the label for basis vectors such that $(1, 0, \cdots, 0)$ in order to figure out whether first variable is in parity function or not and so on. So, we try to write basis vector $(1, 0, \cdots, 0)$ for first variable as the sum of a small number of training examples. By observation, if s is approximately small, we will get the correct label with high confidence. We are agoing to looking for the basis vector as the sum of $\frac{n}{\log n}$ examples since the probability that the sum of label is the correct label is roughly $\frac{1}{2} + \frac{1}{2^{\frac{n}{\log n}}}$, it's small enough. This process can be further repeated to determine the remaining bits like $(0, 1, 0, \cdots, 0)$.

To describe the algorithm, it is convenient to view each example $x \in (0, 1)^n$ as a string of a blocks, each of length b.

Definition : Let V_i be the subspace of $(0, 1)^n$ where last i blocks is zero.
Let i-sample be a set of vectors in V_i uniformly and independently distributed over V_i.

Lemma : Assume we are given an i-sample of size s. We can in time $O(s)$ construct an $(i+1)$-sample of size at most $s - 2^b$ such that each vector in the $(i+1)$-sample is written as the sum of two vectors in the given i-sample.

Proof: Partition all elements in V_i according to values in last non-zero block. This partition has at most 2^b grouping. For each grouping, randomly pick a vector v' and add it to each of the other vectors in its group. and then remove v'. The result is a collection of vectors $u_1, \cdots, u_{s'}$ where $s' \geq s - 2^b$ since we remove at most one vector per class.

Each u_i is formed by summing two vectors in V_i which has identical component in last non-zero block, zeroing out that block. Therefore u_i is in V_{i+1}. In addition, each u_i is an independent, uniform-random member of V_{i+1} since each u_i is formed by summing randomly chosen vectors in V_i. ■

Recall that we try to write basis vector $(1, 0, \cdots, 0)$ as the sum of $\frac{n}{\log n}$ vectors. Apply Lemma $a - 1$ times to construct $(a-1)$-samples. Then there are some vectors, whose fist block is random b bits and the rest of blocks are all zero. Each vector is the sum of 2^{a-1} of our labeled example. Note that V_{a-1} contains only 2^b distinct vectors. If $(1, 0, \cdots, 0)$ does not occur among our examples, we repeat the process with new labeled examples. Note that te expected number of repetitions is only constant. If it occur, the propability that the vector is $(1, 0, \cdots, 0)$ is $\frac{1}{2^b}$.

Next, assume we got $(1, 0, \cdots, 0)$. By observation the sum of labels is correct label with probability $\frac{1}{2} + \frac{1}{2}(1-2\eta)^{2^{a-1}}$. If we want to be sure that we actually get $(1, 0, \cdots, 0)$ with reasonable probability, we need to repeat this process 2^b. This process give us the label error with probability $(\frac{1}{1-2\eta})^{2^a}$. If we choose $a = \frac{1}{2}\log n$ and $b = \frac{2n}{\log n}$, the run time is $2^{\frac{n}{\log n}}$ with small error.

References

[1] Avrim Blum, Adam Kalai and Hal Wasserman. Noise-tolerant learning, the parity problem, and the Statistical Query model. JACM 50(4):506-519(2003). Extended abstract in STOC'00,pp.435–440