CS 270 - Homework 3

Walid Krichene (23265217)

(1) Consider the randomized multiplicative weights algorithm on the alternatingly correct pair of experts A, B, with update $(1-\epsilon)$. Without loss of generality, assume that A is correct on even days, and B on odd days. At iteration t, the loss vector $l(t) = [l_A(t), l_B(t)]$ is given by

$$l(t) = \begin{cases} [1,0] & \text{if } t \text{ is odd} \\ [0,1] & \text{if } t \text{ is even} \end{cases}$$

therefore the wight updates are given by

$$w(t+1) = \begin{cases} [w_A(t)(1-\epsilon), w_B(t)] & \text{if } t \text{ is odd} \\ [w_A(t), w_B(t)(1-\epsilon)] & \text{if } t \text{ is even} \end{cases}$$

by induction, stating from the uniform weights $w(1) = [1,1]$, we have

$$w(2k+1) = [(1-\epsilon)^k, (1-\epsilon)^k]$$
$$w(2k) = [(1-\epsilon)^k, (1-\epsilon)^{k-1}]$$

Let $L(t)$ be the (random) loss incurred by the algorithm at iteration t. Then the expected loss at iteration t is

$$\mathrm{E}[L(t)] = \frac{w_A(t)l_A(t) + w_B(t)l_B(t)}{w_A(t) + w_B(t)} = \begin{cases} \frac{(1-\epsilon)^k}{2(1-\epsilon)^k} = \frac{1}{2} & \text{if } t = 2k+1 \\ \frac{(1-\epsilon)^{k-1}}{(1-\epsilon)^k + (1-\epsilon)^{k-1}} = \frac{1}{2-\epsilon} & \text{if } t = 2k \end{cases}$$

Finally, the expected loss over T iterations is

$$\sum_{t=1}^{T} \mathrm{E}[L(t)] = \lceil T/2 \rceil \frac{1}{2} + \lfloor T/2 \rfloor \frac{1}{2-\epsilon}$$

since $1/2 \leq 1/(2-\epsilon)$, an upper bound on the expected loss is simply given by

$$\sum_{t=1}^{T} \mathrm{E}[L(t)] \leq \frac{T}{2-\epsilon}$$

For the case $\epsilon = \frac{1}{2}$, the expectation becomes

$$\sum_{t=1}^{T} \mathrm{E}[L(t)] = \lceil T/2 \rceil \frac{1}{2} + \lfloor T/2 \rfloor \frac{2}{3}$$

(2) The original matrix of the game is

$$A = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & -1 & -1 \end{bmatrix}$$

In order to have payoffs in $[0, 1]$, an equivalent game is given by the scaled and translated matrix $\tilde{A}$, given by $\tilde{A}_{i,j} = \frac{A_{i,j}+1}{2}$

$$\tilde{A} = \begin{bmatrix} 1/2 & 1 & 0 \\ 0 & 1/2 & 1 \\ 1 & 0 & 1/2 \\ 1/2 & 0 & 0 \end{bmatrix}$$

The experts algorithm applied to the 2-player zero sum game proceeds as follows:

- the row player applies the expert's algorithm, and maintains the probability distribution $p(t)$.
- the outcomes at iteration t are given by

$$l(t) = A.y^T(t)$$

where $y(t)$ is the best response to $p(t)$ (the column player is assumed to play optimally), given by

$$y(t) = \arg\max_{y \in \Delta} p(t) A y^T$$

the column player simply picks $\arg\max_j (p(t)A)_j$, and picks the lowest such j in case of ties.

The results are

T	$\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x(t)$	$\delta/2$
10	(0.24921, 0.30276, 0.19881, 0.24921)	0.2887
100	(0.19911, 0.32595, 0.06973, 0.40520)	0.0627
$\lceil 100 \ln 4 \rceil$	(0.18886, 0.32753, 0.05072, 0.43289)	0.0489

where $\delta/2$ is computed using the average distributions, and is the maximum increase in player utility if she unilaterally changes her strategy. In other words

$$\delta/2 = \max(\bar{x}\tilde{A}\bar{y}^T - R(\bar{y}), C(\bar{x}) - \bar{x}\tilde{A}\bar{y}^T)$$

where

$$\bar{x} = \frac{1}{T}\sum_{t=1}^{T} x(t)$$

$$\bar{y} = \frac{1}{T}\sum_{t=1}^{T} y(t)$$

and

$$R(y) = \min_x x\tilde{A}y^T$$

$$C(x) = \max_y x\tilde{A}y^T$$

```scala
package experts
import util.linAlgebra._

class InvalidArgumentException(message: String) extends Exception(message)

trait Nature {
  def update(strategy: Array[Double])
}

abstract class Expert[N<:Nature](val nature: N) {
  def nextLoss(): Double
}

class ZeroSumGame(val A: Array[Array[Double]]) extends Nature {
  // A is the payoff matrix, assumed to be normalized (entries in [0, 1])
  // row minimizes, column maximizes
  val nRows = A.length
  val nCols = A(0).length
  var rounds = 0
  var bestColResponse = 0

  val cumulativeColStrategy = new Array[Double](nCols)
  val cumulativeRowStrategy = new Array[Double](nRows)

  private def getNormalized(cumulativeStrategy: Array[Double]): Array[Double] = {
    val norm = cumulativeStrategy.sum
    cumulativeStrategy.map(_/norm)
  }

  def getAvgColStrategy() = getNormalized(cumulativeColStrategy)
  def getAvgRowStrategy() = getNormalized(cumulativeRowStrategy)

  def computeOutcome(x: Array[Double], y: Array[Double]): Double = {
    x.times(A).times(y.transp()).doubleValue
  }

  def computeBestColResponse(rowStrategy: Array[Double]): (Int, Double) = {
    // compute the best column response
    val ((_, argmax), max) = rowStrategy.times(A).argMax()
    (argmax, max)
  }

  def computeBestRowResponse(colStrategy: Array[Double]): (Int, Double) = {
    val ((_, argmin), min) = A.times(colStrategy.transp()).argMin()
    (argmin, min)
  }

  def update(rowStrategy: Array[Double]) {
    bestColResponse = computeBestColResponse(rowStrategy)._1
    rounds += 1
    // update the average strategies
    for(i <- 0 to nRows-1)
      cumulativeRowStrategy(i) += rowStrategy(i)

    cumulativeColStrategy(bestColResponse) += 1
  }
}

class ZeroSumGameExpert(game: ZeroSumGame, row: Int) extends Expert[ZeroSumGame](game){
  def nextLoss(): Double = {
    val colResponse = game.bestColResponse
    return game.A(row)(colResponse)
  }
}
```

```
65
66 // Experts algorithm
67 // Type parameter N must extend Nature, and experts are of type Expert[N]
68 // (the experts share an instance of Nature)
69 // nature keeps extra information specific to the particular game being played.
70 // for example, if nature is a ZeroSumGame, it provides methods to compute best
71 // column response, and it keeps track of the average row and column strategies.
72 // In the routing game, it would provide a best response method that computes the
73 // shortest path, etc.
74 class ExpertAlgorithm[N<:Nature](epsilon: Double, experts: List[Expert[N]]) {
75   if(epsilon >= 1 || epsilon <= 0)
76     throw new InvalidArgumentException("epsilon should be in (0,1)")
77
78   val nature = experts(0).nature
79   val support = experts.length
80   var strategy = Array.fill(support)(1./support)
81
82   def normalize() {
83     val norm = strategy.sum
84     for(i<- 0 to support-1)
85       strategy(i)/=norm
86   }
87
88   def next() {
89     nature.update(strategy)
90     val weights = experts.map(_.nextLoss).map(math.pow(1-epsilon, _)).toArray
91     strategy = strategy.dotMult(weights).arrayValue()
92     normalize()
93   }
94 }
```

```
1 package experts
2
3 object main {
4   def main(args: Array[String]): Unit = {
5     val epsilon = .1
6     // payoff matrix
7     val A = Array[Array[Double]](
8       Array[Double](.5, 1, 0),
9       Array[Double](0, .5, 1),
10      Array[Double](1, 0, .5),
11      Array[Double](.5, 0, 0)
12    )
13    val game = new ZeroSumGame(A)
14    val experts = (0 to 3).map(new ZeroSumGameExpert(game, _)).toList
15    val alg = new ExpertAlgorithm[ZeroSumGame](epsilon, experts)
16
17    for(t <- 1 to math.ceil(100*math.log(4)).intValue)
18      alg.next()
19
20    val y = game.getAvgColStrategy()
21    val x = game.getAvgRowStrategy()
22    val delta = math.max(
23          game.computeBestColResponse(x)._2 - game.computeOutcome(x, y),
24          game.computeOutcome(x, y) - game.computeBestRowResponse(y)._2)
25
26    println(game.rounds)
27    println(x.toList)
28    println(delta)
29  }
30 }
```

The code is in the files *experts.scala* and *main.scala*, and some helper classes are in *linAlgebra.scala*.

(3) Consider the fractional routing problem on the graph $G = (V, E)$, with pairs (s_i, t_i), $i \in [k]$. Let $|E| = m$ and $|V| = n$. Find a $1 + \epsilon$ approximate solution to the fractional path routing problem in $O(km\frac{\log_2 m}{\epsilon^2})$ time (with high probability).

Lemma 1 *If one chooses uniformly at random from k possibilities, T times, with $T \geq \frac{8k \ln k}{\epsilon^2}$, then for each $i \in [k]$, if T_i is the number of times i is chosen, we have*

$$\mathrm{P}\left(T_i \geq \frac{T}{k}(1-\epsilon)\right) > 1 - \frac{1}{k}$$

Lemma 2 *If $G = \sum_{t=1}^{T} G_t$, where the choice of G_t is made independently of others, and $T \geq \frac{8k \ln k}{\epsilon^2}$, and G_t have mean and covariance at most k, then*

$$\mathrm{P}(G \leq \mathrm{E}[G](1+\epsilon)) > 1 - \frac{1}{k}$$

answer Consider the following randomized version of the experts algorithm, where the toll player maintains a probability distribution $p_e(t), e \in E$.

Algorithm 1 Experts algorithm

Initialize $T_i := 0 \ \forall i$

for each iteration $t \in \{1, \ldots, T\}$ **do**

the routing player chooses, uniformly at random, $i \in [k]$, and routes (s_i, t_i) along the shortest path in the metric given by $p_e(t)$

let $f_i(t) \in F_i$ be the corresponding flow solution (here F_i is the set of feasible flows that connect (s_i, t_i)).

let $f_j(t) = 0$ for all $j \neq i$

increment T_i

the toll player updates the distribution using the experts update rule

$$p_e(t+1) \propto p_e(t)(1+\epsilon)^{g_e(t)}$$

where the gain of expert e on day t is the congestion of edge e under flow $f(t)$

$$g_e(t) = c(e, f(t))$$

Note: since the routing player only routes one pair (s_i, t_i) on each day, the gains are at most one: $g_e(t) \leq 1$ for all e and t

end for

the candidate approximate flow solution is

$$\bar{f} = \sum_{i=1}^{k} \frac{1}{T_i} \sum_{t=1}^{T} f_i(t)$$

($\bar{f}$ is a random variable)

Let

- $c^{max}(\bar{f}) = \max_{e \in E} c(e, \bar{f})$ be the maximal congestion corresponding to $\bar{f}$ (random variable)
- c^* be the value of the toll congestion game

$$c^* = \min_{f \in F_{[k]}} \max_{p \in \Delta_E} c(p, f) = \max_{p \in \Delta_E} \min_{f \in F_{[k]}} c(p, f)$$

where p is in the set of probability distributions over the set of edges E, and f is in the set of feasible flows that connect all k pairs ($F_{[k]} = F_1 + \cdots + F_k = \{f_1 + \cdots + f_k, f_i \in F_i\}$).
Note: for each fixed p, the congestion function $f \mapsto c(p, f)$ is linear. This is crucial for the proof.

- G be the expected total gain

$$G = \sum_{t=1}^{T} c(p(t), f(t))$$

(random variable since the $f(t)$'s are random). $c(p(t), f(t))$ is the cost of the shortest path picked on day t.

- G^* be the gain of the best expert (random variable)

$$G^* = \max_e \sum_t c(e, f(t))$$

and assume we play the game for $T = \frac{8k \log(m)}{\epsilon^2}$ days. The complexity of each iteration is $O(m \log(m))$ and corresponds to computing one shortest path for the randomly chosen pair. Thus the total time complexity is $O(Tm \log(m)) = O(km \frac{\log(m)^2}{\epsilon^2})$. The claim is that with high probability, $\bar{f}$ is a $(1 + 3\epsilon)$-approximate solution, i.e.

$$1 \leq \frac{c^{\max}(\bar{f})}{c^*} \leq (1 + 3\epsilon)$$

(here $c^{\max}(\bar{f})$ is greater than the value of the game since the toll player plays second) or equivalently

$$\frac{c^*}{c^{\max}(\bar{f})} \geq (1 - 3\epsilon)$$

Proof of claim: we use a sequence of inequalities:

1. by the experts algorithm analysis, we have that for every realization, $G \geq (1 - \epsilon)G^* - \log(m)/\epsilon$, thus using $T = 8k \log m/\epsilon^2$, we have

$$\frac{kG}{T} \geq (1 - \epsilon)\frac{kG^*}{T} - \frac{k \log(m)}{\epsilon T} = (1 - \epsilon)\frac{kG^*}{T} - \frac{\epsilon}{8} \tag{1}$$

2. with high probability (probability $\geq 1 - \frac{1}{k}$),

$$\frac{kG^*}{T} \geq (1 - \epsilon)c^{\max}(\bar{f}) \tag{2}$$

Indeed: by definition, $G^* = \max_e \sum_t c(e, f(t))$. Fixing e, we can write

$$\begin{aligned}
\sum_t c(e, f(t)) &= \sum_t \sum_i c(e, f_i(t)) \\
&= \sum_i T_i \frac{1}{T_i} \sum_t c(e, f_i(t)) \\
&\geq (1 - \epsilon)\frac{T}{k} \sum_i \frac{1}{T_i} \sum_t c(e, f_i(t)) && \text{using } T_i \geq \frac{T}{k}(1 - \epsilon) \text{ (Lemma 1)} \\
&= (1 - \epsilon)\frac{T}{k} c(e, \bar{f}) && \text{by linearity of } c(e, .)
\end{aligned}$$

taking the maximum over e, we have $\frac{k}{T} \max_e \sum_t c(e, f(t)) \geq (1 - \epsilon) \max_e c(e, \bar{f})$, i.e. $\frac{k}{T} G^* \geq (1 - \epsilon)c^{\max}(\bar{f})$

3. with high probability (probability $\geq 1 - \frac{1}{k}$),

$$c^* \geq (1-\epsilon)\frac{k}{T}G \tag{3}$$

Indeed, we have $G = \sum_{t=1}^{T} c(p(t), f(t))$, where $f(t) = f_i(t)$ for some i, and the choice of i is made independently and uniformly at random. We have

$$\mathrm{E}[c(p(t), f(t))] = \frac{1}{k} c(p(t), k\,\mathrm{E}[f(t)])$$

by linearity. Here we observe that $\mathrm{E}[f(t)] = \sum_{i=1}^{k} \frac{1}{k} f_i(t)$ (each i is chosen with probability $1/k$), therefore $k\,\mathrm{E}[f(t)] = \sum_{i=1}^{k} f_i(t)$ is the solution to the k-shortest paths, routing all pairs (since each $f_i(t)$ is the shortest path routing pair i). In other words, $c(p(t), k\,\mathrm{E}[f(t)]) = \min_{f \in F_{[k]}} c(p(t), f)$, and thus is less than c^* ($c^* = \max_p \min_f$). Now

$$c^* \geq c(p(t), \mathrm{E}[f(t)]) = k\,\mathrm{E}[c(p(t), f(t))]$$

(where each $c(p(t), f(t))$ has expectation and variance less than one). Summing over t,

$$Tc^* \geq k\,\mathrm{E}[G]$$

then using Lemma2, we have with high probability, $\mathrm{E}[G] \geq G(1-\epsilon)$, therefore

$$c^* \geq \frac{k}{T}\,\mathrm{E}[G] \geq G(1-\epsilon)$$

Combining these inequalities, we have with high probability

$$\begin{aligned}
c^* &\geq (1-\epsilon)\frac{kG}{T} && \text{by (3)} \\
&\geq (1-\epsilon)((1-\epsilon)\frac{kG^*}{T} - \frac{\epsilon}{8}) && \text{by (1)} \\
&\approx (1-\epsilon)^2\frac{kG^*}{T} - \frac{\epsilon}{8} \\
&\geq (1-\epsilon)^3 c^{\max}(\bar{f}) - \frac{\epsilon}{8} && \text{by (2)} \\
&\approx (1-3\epsilon)c^{\max}(\bar{f}) - \frac{\epsilon}{8}
\end{aligned}$$

(there is a $\epsilon/8$ additive term here, but I assume it is okay since we had a similar term in the original algorithm)

(4) Consider investing the following scenario. There are n possible instruments one can invest in in a period of T time intervals, where each instrument either pays one of 0 dollars or 1 dollar at each time. You have inside information that at least one of the instruments yields P dollars over the T time intervals. One can only invest in one instrument. The problem is to decide which one to invest in and when to do so.

(a) Show that any deterministic algorithm may yield zero dollars (unless $n = 1$)

proof Let us denote the gain of instrument i on time t by $g_i(t)$. Consider a deterministic algorithm A, and consider the sequence $g^{perfect}$ of gains such that $g_i^{perfect}(t) = 1\ \forall i$ and $\forall t$. Let i_0, t_0 be, respectively, the instrument invested in and the time invested in, when A is run on $g^{perfect}$. Since A is deterministic (and causal!) when we run A on any sequence g such that $g_i(t) = 1\ \forall i$ and $\forall t < t_0$, the algorithm will invest in i_0 on t_0. In particular if run on g^{haha} defined by

$$g_i^{haha}(t) = \begin{cases} 0 & \text{if } i = i_0 \text{ and } t \geq t_0 \\ 1 & \text{otherwise} \end{cases}$$

the gain is 0. (note that the sequence is feasible since all instruments except i_0 have maximal gain)