Trabalho 1 - Algoritmos e Incerteza
Entrega: 9 de Outubro

Marco Molinaro

Teoria

Problema 1

Considere o problema **Load Balancing** no modelo online. Existem m máquinas e n jobs que tem que ser processados em uma das maquinas. O tempo de processamento dos jobs depende da maquina atribuída; usamos $p_i^t \geq 0$ para denotar o tempo que o job t leva caso atribuído na maquina i. A *carga de uma máquina* é a soma dos tempos de processamento os jobs atribuidos a ela.

Jobs chegam um a um, ou seja, recebemos a sequencia de vetores $p^1, p^2, \ldots, p^n$ item a item, e quando um job chega ele tem que ser atribuído a uma das máquinas (somente com o conhecimento dos jobs vistos ate agora). O objetivo é minimizar a maior carga das máquinas (i.e. queremos que todas as máquinas tenham carga "pequena") ao fim desse processo.

Ou seja, escolhe online vetores $x^1, \ldots, x^n \in \{0,1\}^m$ satisfazendo $\sum_i x_i^t = 1$ (então $x_i^t = 1$ é equivalente a atribuir job t à máquina i) de forma a minimizar

$$\|p^1 \star x^1 + \ldots + p^n \star x^n\|_{\max},$$

onde $u \star v$ é multiplicação coordenada a coordenada (i.e., $(u \star v)_i = u_i v_i$) e $\|v\|_{\max} = \max_i v_i$.

1. Considere o seguinte algoritmo guloso: Seja $\ell_1^{t-1}, \ldots, \ell_m^{t-1}$ as cargas nas máquinas até o tempo t; o guloso atribui o job do tempo t de forma a minimizar a maior carga das máquinas após a atribuição (i.e. escolhe x^t tal que $\|\ell^{t-1} + p^t \star x^t\|_{\max}$ seja mínimo).

   Mostre que pra todo $m \geq 2$ esse algoritmo guloso **não** é $(m-1)$-competitivo.

2. Considere o caso em que cada job tem o mesmo tempo de processamento em todas a máquinas (mas o jobs podem ter tempos de processamento diferentes). Mostre que nesse caso o algoritmo guloso acima é 2-competitivo.

   Dica: Considere o "volume" da instancia, i.e. a soma dos tempos de processamento de todos os jobs em todas as máquinas. Qual é o minimo custo que OPT paga dado um tal volume da instancia? Baseado na análise do último job, dê uma cota inferior para o volume com relação ao custo do algoritmo. Ponha essas cotas juntas para obter o resultado.

Problema 2

Considere o seguinte problema online: *Problema da Contratação* (*Adversarial Secretary Problem*). Uma instância desse problema consiste de uma sequencia de número **distintos** $a_1, a_2, \ldots, a_n$ (modelando qualidade de candidatos a um emprego), reveladas ao longo do tempo (sendo entrevistados). No tempo t, o número a_t é revelado. Apenas com a informação $a_1, a_2, \ldots, a_t$ até o momento, o algoritmo (tipicamente aleatório) tem que decidir selecionar (contratar) o número a_t ou não. O algoritmo só pode selecionar um número durante o jogo todo, e não pode mudar de ideia uma vez que selecionar um número; ou seja, o jogo termina assim que o algoritmo selecionar o primeiro número. O objetivo é maximizar a probabilidade (em relação à aleatorização do algoritmo) do algoritmo selecionar o **máximo** da sequência $\max_t a_t$ (melhor candidato). O *valor* de um algoritmo é essa probabilidade.

O valor ótimo offline desse problema é 1 (i.e., com probabilidade 1 a solução ótima seleciona o maior número). Utilizando o Princípio minimax de Yao, mostre que nenhum algoritmo aleatorizado tem competitividade melhor do que $\frac{1}{n}$, i.e., tem valor maior do que $\frac{1}{n}$.

Dica: Considere instâncias da forma $(1, 2, \ldots, i, 0, \ldots, 0)$. Considere também a execução de um algoritmo determinístico fixo na instância $(1, 2, \ldots, n)$.

Prática

(Exercício 10.3 do livro "Introduction to Online Convex Optimization" do Elad Hazan.)

(Reconhecimento de caracteres usando boosting.) Baixe o training set do "MNIST database"; pode baixar em qualquer formato que encontrar, alguns examplos são:

- `http://deeplearning.net/tutorial/gettingstarted.html`
- `https://pjreddie.com/projects/mnist-in-csv/`
- `http://cis.jhu.edu/~sachin/digit/digit.html`
- `http://scikit-learn.org/stable/datasets/`
- `https://github.com/mrgloom/MNIST-dataset-in-different-formats`

Para simplificar, escolha um par de digitos para tentar reconhecer/distinguir; chamamos esse dígitos de a e b.

Primeiro, para cada um dos 28×28 pixels, crie um classificador para distinguir dígitos a e b **apenas usando esse pixel**. Depois, utilize o algoritmo de boosting baseado no Multiplicative Weights Update para combinar esse classificadores single-pixel em um classificador melhor.

Reporte brevemente a qualidade dos classificadores single-pixel e a qualidade do classificador final. Comente quantas iterações no boosting você usou. Faça um gráfico mostrando a qualidade do classificador combinado com o passar das iterações.

Comente se a hipótese da existência de weak-learner dentro dos seus classificadores single-pixel é satisfeita. Reporte também o peso que o boosting deu a cada um dos classificadores single-pixel para composição do classificador final.