Optimización Numérica: Laboratorio 1

rodrigo.mendoza@itam.mx

3 Septiembre 2019

1 Introducción a Python

En este laboratorio vamos a implementar algunos algoritmos simples para resolver el problema de regresión lineal. El objetivo principal de este laboratorio es adquirir experiencia con `numpy` y `python`. La mejor manera de aprender `python` (y cualquier otro lenguaje de programación) es programando cosas sencillas y buscando en Google o StackOverflow cada vez que encuentres un problema. Si es necesario, revisar los siguientes tutoriales:

1. `python`: http://web.stanford.edu/class/cs224n/readings/python-review.pdf
2. `numpy`: http://cs231n.github.io/python-numpy-tutorial/

2 Regresión

Supongamos que tenemos un conjunto de datos $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \mathbb{R} : i \in [m]\}$. Asumimos un modelo de la forma $y_i \sim \boldsymbol{\beta}^T \mathbf{x}_i + \varepsilon_i$ donde ε_i es un error de medición. El método problema de mínimos cuadrados ajusta este modelo resolviendo:

$$\min_{\boldsymbol{\beta}} \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|^2 \tag{1}$$

1. Encuentre el estimador $\hat{\boldsymbol{\beta}}$ de mínimos cuadrados de manera analítica.
2. Programe el estimador $\hat{\boldsymbol{\beta}}$ de mínimos cuadrados que encontró en la pregunta anterior. *Sólo se permiten usar librerías estándar de Python,* `numpy`*, y* `scipy.linalg` *excepto* `numpy.linalg.lstsq`

```
def mc(X, y):
    '''(X,y):  Datos de entrenamiento [X.shape=(m,p), y.shape=(m,)]
               X,y matrices de numpy
    '''
    ...
    return beta
```

3. ¿Cuál es la complejidad computacional de su algoritmo?
4. Los problemas de aprendizaje de máquina suelen formularse como un problema de minimización de la *función de riesgo empírica*. En mínimos cuadrados tiene la forma de:

$$\mathcal{L}^{emp}(\boldsymbol{\beta}) = \frac{1}{m} \sum_{i=1}^{m} \ell(\boldsymbol{\beta}^T \mathbf{x}_i, y_i) \tag{2}$$

   ¿Qué forma tiene $\ell(\cdot, \cdot)$ en el problema de mínimos cuadrados?

5. La función de riesgo empírica puede ser minimizada por medio del *método del gradiente*:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \nabla \mathcal{L}^{emp}(\boldsymbol{\beta}) \tag{3}$$

   A $\eta > 0$ se le llama la *tasa de aprendizaje*. Implemente el método del gradiente en Python para el problema de mínimos cuadrados. Introduzca la variable `eps` para denotar la tolerancia de convergencia. ¿Qué condición de convergencia propondría?

```python
def metodo_gradiente(X, y, eta, eps):
    '''
    (X,y):  Datos de entrenamiento [X.shape=(m,p), y.shape=(m,)]
            X,y matrices de numpy
    eta:    Tasa de aprendizaje [real]
    eps:    Tolerancia de convergencia [real]
    '''
    ...
    return beta
```

6. Una variante popular del método del gradiente consiste en usar *un dato* por actualización de $\boldsymbol{\beta}$ y repetir para todo el conjunto de datos. Esto es,

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \nabla \ell(\boldsymbol{\beta}^T \mathbf{x}_i, y_i), \quad \text{para } i \in [m]. \tag{4}$$

A esta variante se le conoce como el *método estocástico del gradiente*. Implemente esta función en Python. En la implementación, reordene los datos de manera aleatoria al inicio de cada iteración (4).

```python
def sgd(X, y, eta, eps):
    '''
    (X,y):  Datos de entrenamiento [X.shape=(m,p), y.shape=(m,)]
            X,y matrices de numpy
    eta:    Tasa de aprendizaje [real]
    eps:    Tolerancia de convergencia [real]
    '''
    ...
    return beta
```

7. Otra variante es el *método estástico del gradiente con mini-lotes*. En esta variante se hace la actualización

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \frac{1}{|S|} \sum_{j \in S} \nabla \ell(\boldsymbol{\beta}^T \mathbf{x}_i, y_i), \quad \text{para mini-lote } S \in \{S_1, \ldots, S_q\}. \tag{5}$$

Generalice su función `sgd` para que pueda trabajar con *mini-lotes* de tamaño t. Esta función también generaliza el método del gradiente cuando $t = m$.

```python
def sgd(X, y, eta, eps, t):
    '''
    (X,y):  Datos de entrenamiento [X.shape=(m,p), y.shape=(m,)]
            X,y matrices de numpy
    eta:    Tasa de aprendizaje [real]
    eps:    Tolerancia  de convergencia [real]
    t:      Tamano del mini-lote [entero en [1,m]]
    '''
    ...
    return beta
```

8. Cree una función `datos(m,p)` que genere datos $(\mathbf{X}, \mathbf{y}) \in \mathbb{R}^{m \times p} \times \mathbb{R}^m$ con $X_{i,j} \sim \mathcal{N}(0,1)$ y $y_i \sim \mathcal{N}(3,2)$.

9. Diseñe un experimento numérico para elegir la tasa de aprendizaje η.

10. Fije $p = 30$, grafique el tiempo de ejecución de sus algoritmos para $m \in [30, 40, 50, ..., 200]$. Puede medir el tiempo de ejecución de su código usando la siguiente fórmula:

```python
import time
t0 = time.time()
beta = f(X,y)
t1 = time.time()
tiempo = t1 - t0
```

11. Fije $p = 30$ y $m = 1000$. Grafique el error de estimación de `sgd` $\|\hat{\boldsymbol{\beta}}^{(iter)} - \hat{\boldsymbol{\beta}}^{MC}\|_2$ por iteración para $t = [1, \frac{m}{4}, \frac{m}{2}, \frac{3m}{4}, m]$.