Computational statistics - TP 1

LACOMBE Yoach

Exercice 3

question 1

Il s'agit de faire une descente de gradient stochastique (SGD) pour minimiser le risque empirique. Ici le risque empirique est :

$$R_n(w) = \sum_{i=0}^{n-1} (y_i - w^T x_i)^2/n = \sum_{i=0}^{n-1} j(w, z_i)/n$$

Décrivons rapidement l'algo de SGD:

1. w_0 donné
2. k -> k+1, $w_{k+1} = w_k - \epsilon_k \nabla_w j(w_k, z_{k+1})$

Il suffit donc maintenant de calculer le gradient de j par rapport à w et de décider du pas ϵ_k.

On prend $\epsilon_k = 1/k^\alpha$ avec α dans]0.5,1].

Le gradien est très facile à calculer, $\nabla_w j(w_k, z_{k+1}) = -2 * (y_{k+1} - w_k^T x_{k+1}) x_{k+1}$.

Ainsi, l'étape d'update revient à :

$$w_{k+1} = w_k + 2/k^\alpha * (y_{k+1} - w_k^T x_{k+1}) x_{k+1}$$
$$w_{k+1}^T = w_k^T + 2/k^\alpha * (y_{k+1} - w_k^T x_{k+1}) x_{k+1}^T$$

Ecrivons maintenant le code. Je choisis comme critère d'arrêt un nombre d'itérations fixe et un w_0 aléatoire (tiré d'un vecteur gaussien centré réduit).

Remarque 1: On peut ajouter une dimension aux samples, $z = [z^T, 1]^T$, afin d'avoir le biais calculé dans l'algorithme.

Remarque 2: Je prends le paradigme de sklearn, c'est-à-dire un dataset de dimension (n,d), càd $data = [x_0, x_1, \ldots, x_n]^T$

```
import numpy as np

def sgd(df, labels, alpha = 0.7, nb_iterations = 10000):
  df = np.c_[ df, np.ones(len(df))]

  n, d = df.shape

  w_transpose = np.random.multivariate_normal(np.zeros(d), np.identity(d))
  w_transpose = np.expand_dims(w_transpose, axis=0)
  for i in range(nb_iterations):
    j = i%n
```

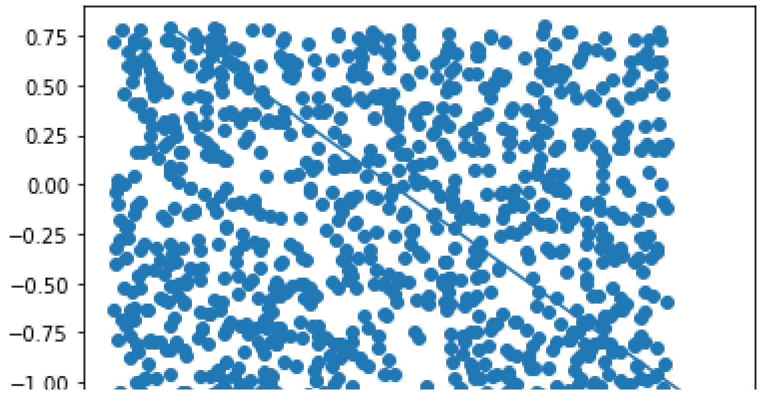
```
    w_transpose = w_transpose + 2*(labels[j] - w_transpose@df[j,:].T)*df[j,:]/((i+1)**alph

  return w_transpose[0,:]
```

question 2

Il faut décider de comment tirer les samples. Pour ça, je décide de tirer les samples d'une loi uniforme $U([u_{min}, u_{max}]^d)$.

```
import matplotlib.pyplot as plt
n = 2000
u_min = -1
u_max = 1
w_true = np.array([5,5, 1])


y_min = (-w_true[0]*u_min - w_true[2])/w_true[1]
y_max = (-w_true[0]*u_max - w_true[2])/w_true[1]
data = (y_max - y_min)*np.random.random_sample((n,2)) + y_min
```

```
plt.scatter(data[:1000,0], data[:1000,1])
plt.plot([u_min,u_max],[y_min, y_max ])
plt.show()

labels = np.zeros(n) - 1

labels[(data@w_true[:2] + w_true[2])>=0] = 1

plt.scatter(data[:1000,0], data[:1000,1], c = labels[:1000])
plt.plot([u_min,u_max], [(-w_true[0]*u_min - w_true[2])/w_true[1], (-w_true[0]*u_max - w_t
plt.show()
```

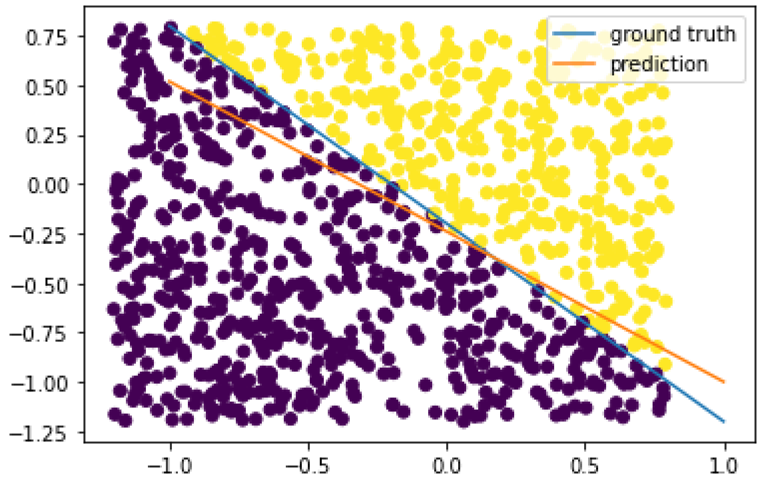


question 3

```
w_pred = sgd(data, labels, alpha  = 0.5, nb_iterations=2000)

print("prediction :", w_pred)
print("true_value :", w_true)
```

```
prediction : [0.86639509 1.14158719 0.27417013]
true_value : [5 5 1]
```

```
plt.scatter(data[:1000,0], data[:1000,1], c = labels[:1000])
plt.plot([u_min,u_max], [(-w_true[0]*u_min - w_true[2])/w_true[1], (-w_true[0]*u_max - w_t
plt.plot([u_min,u_max], [(-w_pred[0]*u_min - w_pred[2])/w_pred[1], (-w_pred[0]*u_max - w_p
plt.legend()
plt.show()
```



Les hyperplans se ressemblent plus ou moins. La méthode paraît donc marcher. On remarque également que le vecteur prédit est plus ou moins le vrai vecteur à une constante multipicative près.

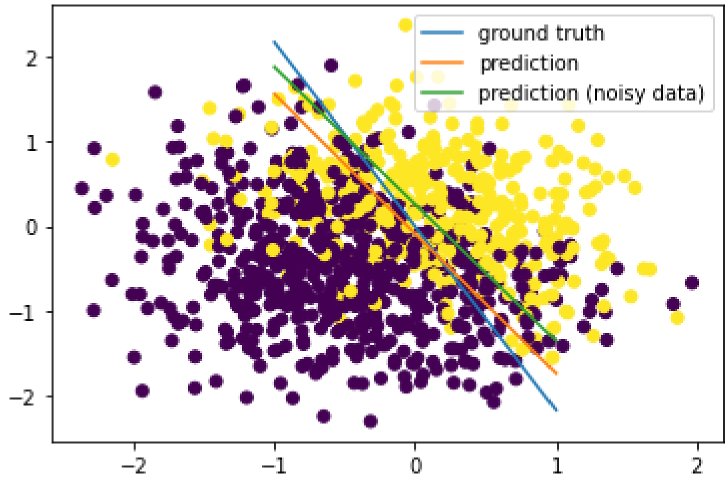
question 4

```
noise = np.random.normal(0, 0.5, size = data.shape)
data_noise = data + noise
w_pred_noise = sgd(data_noise, labels, alpha  = 0.5, nb_iterations=5000)
```

```
print("prediction :", w_pred)
print("prediction (noisy data):", w_pred_noise)
print("true_value :", w_true)
```

```
prediction : [0.86639509 1.14158719 0.27417013]
prediction (noisy data): [ 0.42469729  0.57047459 -0.0637899 ]
true_value : [5 5 1]
```

```
plt.scatter(data_noise[:1000,0], data_noise[:1000,1], c = labels[:1000])
plt.plot([u_min,u_max], [(-w_true[0]*np.min(data_noise[:1000,0]) - w_true[2])/w_true[1], (
plt.plot([u_min,u_max], [(-w_pred[0]*np.min(data_noise[:1000,0]) - w_pred[2])/w_pred[1], (
plt.plot([u_min,u_max], [(-w_pred_noise[0]*np.min(data_noise[:1000,0]) - w_pred_noise[2])/
plt.legend()
plt.show()
```



Ici la prédiction avec noisy data n'a pas l'air d'avoir marché (il faudrait regarder l'accuracy de la prédiction pour en être sur).

Remarque: L'algorithme ne marche pas si les données ne sont pas normalisés (numériquement les valeurs explosent trop vite pour Python). Je normaliserai donc les données pour la prochaine question.

question 5:

```
from sklearn.datasets import load_breast_cancer
```

```
X, y = load_breast_cancer(return_X_y = True)
```

```
from sklearn.preprocessing import normalize
```

```
X_normalized = normalize(X)

y[y==0] = -1


w_decision = sgd(X_normalized, y, alpha  = 0.5, nb_iterations=len(X)*3)
```

Je teste l'algorithme avec l'accuracy (on suppose donc un bias égale à w_decision[-1] -> on aurait pu le choisir afin d'atteindre le trade-off souhaitée entre precision et recall, je ne fais pas ce travail ici).

```
y_pred = np.c_[ X_normalized, np.ones(len(X_normalized))]@w_decision
y_pred = np.sign(y_pred)


print('accuracy', np.mean(y_pred == y))
```

```
    accuracy 0.9068541300527241
```

✓ 0 s terminée à 15:12

Exercice 2 : (3)

Soit $x \notin \{\frac{1}{n}\}_{n\in\mathbb{N}^*}$, f mesurable bornée,

$$\begin{aligned} Pf(x) &= \mathbb{E}[f(X_1) \mid X_0 = x] \quad \text{avec les notations du TD } (U_0 \sim \mathcal{U}([0,1]) \perp\!\!\!\perp X_0) \\ &= \mathbb{E}_{U_0}[f(U_0) \mid X_0 = x] \\ &= \mathbb{E}[f(U_0)] \quad U_0 \perp\!\!\!\perp X_0 \\ &= \int f(t)\pi(t)\,dt \quad \text{avec } \pi \text{ la distribution de } \mathcal{U}([0,1]) \\ &= \text{cste} \end{aligned}$$

Par récurrence, on montre facilement que $\forall n \in \mathbb{N}^*$, $P^n f(x) = \int f(t)\pi(t)\,dt$

car

$$\begin{aligned} P^{n+1}f(x) = P^0[P^n f(x)] &\overset{\text{rom}}{=} \mathbb{E}[P^n f(X_1) \mid X_0 = x] \quad \text{récurrence} \\ &= \mathbb{E}[\underbrace{\int f(t)\pi(t)\,dt}_{\text{cste}} \mid X_0 = x] \\ &= \int f\pi . \end{aligned}$$

(4) Soit $m \geq 2$ et $x = \frac{1}{m}$.

a) ~~$n = 1$, $P^1(x, \frac{1}{m+1}) = \int P(y, \frac{1}{m+1}) P^0(x, dy)$~~

~~Or $P^0(x, dy) = (1 - x^2)\delta_{\frac{1}{m+1}}(dy)$ [car $\mathbb{1}_{A\cap[0,1]} = 0$]~~

~~D'où $P^1(x, \frac{1}{m+1}) = (1 - x^2)\int P(y, \frac{1}{m+1})\delta_{\frac{1}{m+1}}(dy)$~~

Je n'arrive pas à trouver de définitions claires pour P^n (est-ce $P^{n-1} * P$ ou $P * P^{n-1}$?).

J'utilise donc cette déf :

$$P^{n+1}(x, A) = \int P(x, dy) P^n(y, A) \quad \text{et} \quad P^1(x, A) = P(x, A).$$

On a donc pour $n = 1$, $P^1(\frac{1}{m}, \frac{1}{m+1}) = P(\frac{1}{m}, \frac{1}{m+1}) = 1 - \frac{1}{m^2}$.

Par récurrence $\boxed{\forall m \in \mathbb{N}^*, \forall n \in \mathbb{N}^*, \ P^n(\frac{1}{m}, \frac{1}{m+n}) = \prod_{i=0}^{n-1}\left(1 - \frac{1}{(m+i)^2}\right).}$

En effet

$$P^{n+1}\left(\frac{1}{m}, \frac{1}{m+n+1}\right) = \int \underbrace{P(x, dy)}_{=(1-\frac{1}{m^2})\times\delta_{\frac{1}{m+1}}(dy)} P^n\left(y, \frac{1}{m+n+1}\right) = \int\left(1 - \frac{1}{m^2}\right)\times\delta_{\frac{1}{m+1}}(dy) P^n\left(y, \frac{1}{m+n+1}\right)$$

(récurrence)

$$= \left(1 - \frac{1}{m^2}\right) P^n\left(\frac{1}{m+1}, \frac{1}{m+n+1}\right) = \left(1 - \frac{1}{m^2}\right)\times\prod_{i=0}^{n-1}\left(1 - \frac{1}{(m+1+i)^2}\right) = \prod_{i=0}^{n}\left(1 - \frac{1}{(m+i)^2}\right) \quad \square$$

b) Remarquons que $\Pi(A) = \Pi\left(\bigcup_{a\in\mathbb{N}} \left\{\frac{1}{n+1+a}\right\}\right) = 0$ car l'ensemble est de mesure de Lebesgue nul.

Considérons $v_m := p^m\left(\frac{x}{n}, \frac{1}{n+m}\right) = \prod_{i=0}^{m-1}\left(1-\frac{1}{(n+i)^2}\right)$

On a $v_m = \prod_{i=0}^{m-1} \frac{(n+i-1)(n+i+1)}{(n+i)^2} = \underbrace{\prod_{i=0}^{m-1}\frac{n+i-1}{n+i}} \times \underbrace{\prod_{i=0}^{m-1}\frac{n+i+1}{n+i}}$

$= \frac{n-1}{n+m-1} \times \frac{n+m}{n}$

Donc $v_m \sim \left(1-\frac{1}{n}\right)\times\frac{m}{m}$

On a donc $v_m \xrightarrow[m\to+\infty]{} 1-\frac{1}{n}$ ($\neq 0$ pour $n\geq 2$)

Puisque $\forall m\in\mathbb{N}^*$, $a_m := \frac{1}{n+m} \in A$

on a, $p^m(x,A) \geq \underbrace{p^m(x,a_m)}$ $\xrightarrow[m\to+\infty]{} 1-\frac{1}{n} > 0$

Ainsi $p^m(x,A)$ ne CV pas vers 0 donc ne peut pas être CV vers $\Pi(A)$.