TP : Descente de gradient et gradient stochastique

Alain Rakotomamonjy
alain.rakotomamonjy@univ-rouen.fr, sites.google.com/site/alainrakotomamonjy/home

12 novembre 2017

But du TP

Le but de ce TP est de comparer et d'implementer des algorithmes d'apprentissage capable de passer l'echelle des grandes masses de données.

Ex. 1 — Regression par moindre carrées régularisé ℓ_2
Soit le problème d'optimisation suivant

$$\min_{\mathbf{w}} \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

où $\mathbf{X}$ est la matrice des $\{\mathbf{x}_i\}_{i=1}^n$ avec $\mathbf{x}_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$. On pourra par exemple supposer que $\mathbf{y}$ provient de l'observation d'un modèle parcimonieux obtenu suivant le processus suivant.

```
nt = 100;
d = 200;
T = 10;
rsnr=30;
Xt=randn(nt,d);
Xt=Xt./(ones(nt,1)*sqrt(sum(Xt.^2)));

% create signal
ind=randperm(size(Xt,2));
indice=ind(1:T);
weights=randn(T,1);
weights=weights + 1*sign(weights);
yt=Xt(:,indice)*weights;
stdnoise=std(yt)/rsnr;
yt=yt+randn(size(yt)).*(ones(nt,1)*stdnoise);

indapp=1:floor(nt/2);
nb_app = length(indapp);
indval=floor(nt/2)+1:nt;
X=Xt(indapp,:);
y=yt(indapp,:);
Xv=Xt(indval,:);
yv=yt(indval,:);
n=length(indapp);
```

1. Solution analytique
   a) Calculer et implémenter la solution analytique de ce problème.
   b) Quelle est la complexité de la solution obtenue ?
2. Solution du premier ordre
   a) Calculer le gradient de $J(\mathbf{w})$ et implementer une méthode itérative de descent de gradient avec sélection de pas par la règle d'Armijo

```
loss= @(w) 1/2/n*norm(y-X*w).^2;
cost=@(w) loss(w)+lambda*norm(w).^2/2;
lossval = @(w) norm(yv-Xv*w);
costw= cost(w);
tic
```

```
w=randn(d,1);
beta=0.9;
nb_itermax = 10000;
cost_vec= zeros(nb_itermax);
nb_grad = zeros(nb_itermax);

for i=1:nb_itermax
    residu=(y-X*w);
    grad= -1/n*X'*residu+lambda*w;
    stepsize=100;
    costw= cost(w);
    while cost(w - stepsize*grad) > costw -beta*stepsize*grad'*grad;
        stepsize=stepsize/2;
    end;
    w=w-stepsize*grad;
    lossvalGD(i)=lossval(w);
    cost_vec(i) = costw;
    time_vec(i) = toc;
    if i > 1
        nb_grad(i) = nb_grad(i-1)+ nb_app;
    else
        nb_grad(i) = nb_app;
    end
    if norm(grad)<1e-7
        break
    end;
end;
```

b) Quelle est la complexité par itération de cette approche ?

3. Comparaison à une descente de gradient stochastique
   a) Implémenter une méthode de descente de gradient stochastique pour résoudre ce problème

```
tic
ws=randn(d,1);
k=1;
nu=1;
for i=1:2
    indice=randperm(n);
     for j=1:n
        ind=j;
        grad= -X(ind,:)'*(y(ind)-X(ind,:)*ws)+ lambda*ws;
        ws=ws - nu/(1+nu*lambda*k)*grad;
        k=k+1;
        errorSGD(k)=norm(yv-Xv*ws);
        cost_vec_sgd(k) = cost(ws);
        time_vec_sgd(k) = toc;
    end;
end;
```

   b) comparer l'évolution de la fonction de cout de la descente de gradient et du gradient stochastique en fonction du nombre de gradient calculés pour chaque mise à jour. (on tracera $log(f(w) - f(w^\star))$ par exemple pour (1) $n = 1000$, $d = 50$ (2) $n = 10000$, $d = 50$ (3) $n = 100000$, $d = 50$. Il faudra modifier le code pour calculer le nombre de gradient.
   c) Comparer également l'evolution de l'erreur de validation (un affichage en log-log pourrait etre utile ici),
   d) Quelle méthode est plus rapide à atteindre une erreur de validation correcte ?
4. Implementer une méthode de gradient stochastique avec reduction de variance et mener la même comparaison.

```
tic
ws=randn(d,1);
k=1;
nu=1 ;
for i=1:nb_epoch
    residu=(y-X*w);
    grad= -1/n*X'*residu+lambda*w;
    indice=randperm(n);
     for j=1:n
        ind=j;
        grad_ws= -X(ind,:)'*(y(ind)-X(ind,:)*ws)+ lambda*ws;
        grad_w= -X(ind,:)'*(y(ind)-X(ind,:)*w)+ lambda*w;

        ws=ws - nu * ( grad_ws - grad_w + grad);
        lossvalsvrgd(k)=lossval(ws);
        cost_vec_svrgd(k) = cost(ws);
        time_vec_svrgd(k) = toc;
        k=k+1;
    end;
    w = ws;

end;
toc
```

Ex. 2 — Méthode du premier ordre pour la régression logistique

On cherche à apprendre un modèle de classification en utilisant un cout de regression logistique. Le problème d'optimisation qui nous interesse est donc

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp -y_i \mathbf{x}_i^\top \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

On pourra appliquer ce problème aux données disponibles sur Moodle

```
nbapp=10000;
ind=randperm(size(x,1));
X=x(ind,:);
y=y(ind);
indapp=1:nbapp;
indval=nbapp+1:size(X,1);
[n,d]=size(X);
Xa=X(indapp,:);
Xv=X(indval,:);
ya=y(indapp);
yv=y(indval);
meanX=mean(Xa);
Xa=bsxfun(@minus,Xa,meanX);
sumA=sqrt(sum(Xa.^2));
Xa=bsxfun(@rdivide,Xa,sumA);
Xv=bsxfun(@minus,Xv,meanX);
Xv=bsxfun(@rdivide,Xv,sumA);
```

1. Solution du premier ordre
   a) Calculer le gradient de $J(\mathbf{w})$ et implementer une méthode itérative de descente de gradient avec sélection de pas par la règle d'Armijo
2. Optimisation par gradient stochastique.

```
for i=1:nbepoch
    indice=randperm(nbapp);
    for j=1:nbapp
```

```
        ind=indice(j);
        % ICI il faut calculer le gradient
        grad=
        ws=ws - nu/(1+nu*lambda*k)*grad;
        k=k+1;
    end;
end;
```

   a) Quel est la compléxité par itération (mise à jour) ? quelle est la compléxité par epoque ? (une epoque correspond à la visite de l'ensemble des n exemples)
   b) Evaluer la fonction cout et l'erreur de validaton apres chaque mise à jour du gradient

3. Comparaison
   a) Comparer l'évolution de la fonction de coût des deux méthodes pour différentes tailles de données d'apprentissage (en fonction du nombre de gradient calculés)
   b) Faites de même pour l'erreur de validation
   c) Pour une meme erreur de validation, quel gain potentiel en temps de calcul peut on obtenir ?
4. Sélection de modèle
   a) Dans l'approche gradient stochastique, étudier quel est l'impact du choix de pas de descente et de son paramètre ν.