Stochastic Optimization and Automatic Differentiation for Machine Learning

Etude de l'algorithme Stochastic Dual Coordinate Ascent (SDCA)

Vincent Le Meur

13 mai 2018

Table des matières

Introduction

L'objectif de cette étude est de résoudre un problème de Support Vector Machines (SVM) dans le cadre d'une classification binaire.
Cette étude se base sur l'article suivant : "Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization (Shai Shalev-Shwartz,Tong Zhang)"

L'essentiel du code se trouve dans le notebook appellé SDCA-LE-MEUR.ipynb. Ce rapport comporte uniquement le code de définition des algorithmes principaux de l'étude ainsi que les graphs obtenus et leurs interprétations.

Nous allons utiliser les notations suivantes :
- $X_1, .., X_n \in \mathbb{R}^d$ pour les variables explicatives
- $y_1, .., y_n \in \{-1, 1\}$ pour les labels associés
- λ le paramètre de régularisation propre à l'algorithme
- $w \in \mathbb{R}$ le paramètre à optimiser du problème primal
- $\phi_i(a) = max(0, 1 - y_i a)$ la fonction de perte (Hinge loss) associée au SVM étudié

Le problème primal associé à l'optimisation de notre SVM consiste alors avec les notations précédentes à minimiser l'application pénalisée suivante : $P(\omega) = \frac{1}{n}\sum_{i=1}^{n} \phi_i(\omega^T X_i) + \frac{\lambda}{2}||\omega||^2$ (1)

Une approche alternative (Dual Coordinate Ascent) à ce problème est celle qui étudie le problème dual. En prenant le conjugué convexe de notre fonction de perte ϕ précédente définie par $\phi_i^*(u) = max_z(zu - \phi_i(z))$, nous pouvons en effet définir le probème équivalent au précédent qui est de maximiser : $D(\alpha) = \frac{1}{n}\sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2}||\frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i x_i||^2$ où ici $\alpha \in \mathbb{R}^n$ (2)

Nous voyons un lien clair entre les deux problèmes. En effet, soit l'application $\omega(\alpha) = \frac{1}{\lambda n}\sum_{i=1}^{n} \alpha_i x_i$ et α^* la solution du problème dual (2) alors $\omega^* = \omega(\alpha^*)$ est également la solution optimale du problème primal (1)

Nous allons alors étudier une approche utilisant cette dualité (SDCA) et la comparer à une approche de descente de sous Gradient :
- Stochastic Dual Coordinate Ascent (SDCA)
- Primal Estimated subGrAdient SOlver for SVM (PEGASOS)

Mise en place du SDCA

L'algorithme SDCA consiste donc à résoudre de manière ittérative les deux problèmes de la manière suivante :

Procedure SDCA$(\alpha^{(0)})$

Let $w^{(0)} = w(\alpha^{(0)})$
Iterate: for $t = 1, 2, \ldots, T$:
  Randomly pick i
  Find $\Delta\alpha_i$ to maximize $-\phi_i^*(-(\alpha_i^{(t-1)} + \Delta\alpha_i)) - \frac{\lambda n}{2}\|w^{(t-1)} + (\lambda n)^{-1}\Delta\alpha_i x_i\|^2$
  $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta\alpha_i e_i$
  $w^{(t)} \leftarrow w^{(t-1)} + (\lambda n)^{-1}\Delta\alpha_i x_i$
Output (Averaging option):
  Let $\bar{\alpha} = \frac{1}{T-T_0}\sum_{i=T_0+1}^{T}\alpha^{(t-1)}$
  Let $\bar{w} = w(\bar{\alpha}) = \frac{1}{T-T_0}\sum_{i=T_0+1}^{T} w^{(t-1)}$
  return $\bar{w}$
Output (Random option):
  Let $\bar{\alpha} = \alpha^{(t)}$ and $\bar{w} = w^{(t)}$ for some random $t \in T_0 + 1, \ldots, T$
  return $\bar{w}$

L'algorithme comporte deux types de sorties :

- Une sortie moyennée qui correspond à prendre pour ω final une moyenne des dernières itérations (ce nombre d'itérations considérées est fixé par le paramètre T_0)
- Une sortie "aléatoire" ou l'on sélectionne ω final aléatoirement parmi les dernières itérations (toujours fixées par le paramètre T_0)

On cherche à chaque étape à calculer $\Delta\alpha_i$. Il se trouve que pour le problème précis que l'on se fixe (résoudre un SVM avec perte Hinge) , on connaît explicitement cette quantité qui est : $\Delta\alpha_i = max(-1, min(\frac{y_i - x_i^T\omega^{t-1}}{||x_i||^2/(\lambda n)} + \alpha_i^{t-1})) - \alpha_i^{t-1}$

Nous allons mettre en place deux variantes de cet algorithme : SDCA et SCDCAperm. La différence entre ces deux algorithmes est la manière d'itérer. Pour SDCA à chaque itération, on pioche uniformément avec remise un $i \in \{1, .., n\}$ alors que pour SDCAperm on effectue à l'avance une permutation aléatoire de $\{1, ..n\}$ sur laquelle on travaille.

```
def SDCA(nb_epoch,T0,X,y,Lambda,method):
    n=X.shape[0]
    list_alpha=[]
    list_omega=[]
    list_primal=[]
    list_dual=[]
    alpha=np.zeros(n)
    T=nb_epoch*n
    omega = (1/(Lambda*n)) * (X.T.dot(alpha))
    for t in range(1,T):
      i=int(np.random.uniform(0,n))
      delta_alpha=y[i]*max(0,min(1, (1-np.dot(X[i].T,np.dot(omega,y[i])))/(np.linalg.norm(X[
          i])/(Lambda*n)) +alpha[i]*y[i])) -alpha[i]
        alpha[i]=alpha[i]+delta_alpha
        list_alpha.append(alpha)
        omega=omega +(1/(Lambda*n))*delta_alpha*X[i]
        list_omega.append(omega)
        list_primal.append(func_primal(omega,Lambda,n,X,y))
        list_dual.append(func_dual(alpha,Lambda,n,X,y))
  if method=="random":
        k=int(np.random.uniform(T0,T))
        omega_final=list_omega[k]
        alpha_final=list_alpha[k]
    if method=="average":
        omega_final=(1/(T-T0))*sum(list_omega[k] for k in range(T0,T-1))
        alpha_final=(1/(T-T0))*sum(list_alpha[k] for k in range(T0,T-1))
    return(omega_final,list_primal,list_dual,list_alpha)
```

Mise en place du SDCA

```
def SDCA_perm(nb_epoch,T0,X,y,Lambda,method):
    n=X.shape[0]
    list_alpha=[]
    list_omega=[]
    list_primal=[]
    list_dual=[]
    alpha=np.zeros(n)
    T=nb_epoch*n
    omega = (1/(Lambda*n)) * (X.T.dot(alpha))
    for k in range(1,nb_epoch):
        # On fixe une permutation sur {1,..,n}
        perm=permutation(n)
        for i in range(len(perm)):
            # On calcule explicitement le delta_alpha_i
            delta_alpha=y[i]*max(0,min(1, (1-np.dot(X[i].T,np.dot(omega,y[i])))/(np.linalg.
                norm(X[i])/(Lambda*n)) +alpha[i]*y[i])) -alpha[i]
            alpha[i]=alpha[i]+delta_alpha
            list_alpha.append(alpha)
            omega=omega +(1/(Lambda*n))*delta_alpha*X[i]
            list_omega.append(omega)
            list_primal.append(func_primal(omega,Lambda,n,X,y))
            list_dual.append(func_dual(alpha,Lambda,n,X,y))
    if method=="random":
        k=int(np.random.uniform(T0,T))
        omega_final=list_omega[k]
        alpha_final=list_alpha[k]
    if method=="average":
        omega_final=(1/(T-T0))*sum(list_omega[k] for k in range(T0,T-1))
        alpha_final=(1/(T-T0))*sum(list_alpha[k] for k in range(T0,T-1))
    return(omega_final,list_primal,list_dual,list_alpha)
```

Mise en place du SDCAPerm

Mise en place de PEGASOS

Nous allons ici mettre en place l'algorithme "concurrent" Pegasos qui effectue une descente de gradient stochastique sur notre problème primal (1). Nous donnons ci-dessous son "pseudo code" :

INPUT: S, λ, T
INITIALIZE: Set $\mathbf{w}_1 = 0$
FOR $t = 1, 2, \ldots, T$
Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.
Set $\eta_t = \frac{1}{\lambda t}$
If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:
Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$
Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):
Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$
[Optional: $\mathbf{w}_{t+1} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|} \right\} \mathbf{w}_{t+1}$]
OUTPUT: $\mathbf{w}_{T+1}$

Ici encore nous allons donner deux variantes de cet algorithme. Une variante pegasos qui, comme indiqué ci-dessus prends un i à chaque itération uniformément parmi $\{1, .., n\}$, et une variante pegasosperm qui à chaque epoch fixe une permutation aléatoire de $\{1, .., n\}$ et itére sur celle-ci :

```
1  def pegasos(nb_epoch,X,y,Lambda):
2      n=X.shape[0]
3      d=X.shape[1]
4      list_omega=[]
5      list_primal=[]
6      omega =np.ones(d)
7      T=nb_epoch*n
8      for t in range(1,T):
9          i=int(np.random.uniform(0,n))
10         eta_t=1/(Lambda*t)
11         if np.dot(X[i].T,np.dot(omega,y[i]))<1:
12             omega=(1-eta_t*Lambda)*omega + eta_t*y[i]*(X[i])
13             list_omega.append(omega)
14             list_primal.append(func_primal(omega,Lambda,n,X,y))
15         else:
16             omega=(1-eta_t*Lambda)*omega
17             list_omega.append(omega)
18             list_primal.append(func_primal(omega,Lambda,n,X,y))
19     return(omega,list_primal)
```

Mise en place de PEGASOS

```
def pegasos_perm(nb_epoch,X,y,Lambda):
    n=X.shape[0]
    d=X.shape[1]
    list_omega=[]
    list_primal=[]
    omega =np.ones(d)
    T=nb_epoch*n
    t=1
    for k in range(1,nb_epoch):
        perm=permutation(n)
        for i in range(len(perm)):
            eta_t=1/(Lambda*t)
            if np.dot(X[i].T,np.dot(omega,y[i]))<1:
                omega=(1-eta_t*Lambda)*omega + eta_t*y[i]*(X[i])
                list_omega.append(omega)
                list_primal.append(func_primal(omega,Lambda,n,X,y))
            else:
                omega=(1-eta_t*Lambda)*omega
                list_omega.append(omega)
                list_primal.append(func_primal(omega,Lambda,n,X,y))
            t=t+1
    return(omega,list_primal)
```

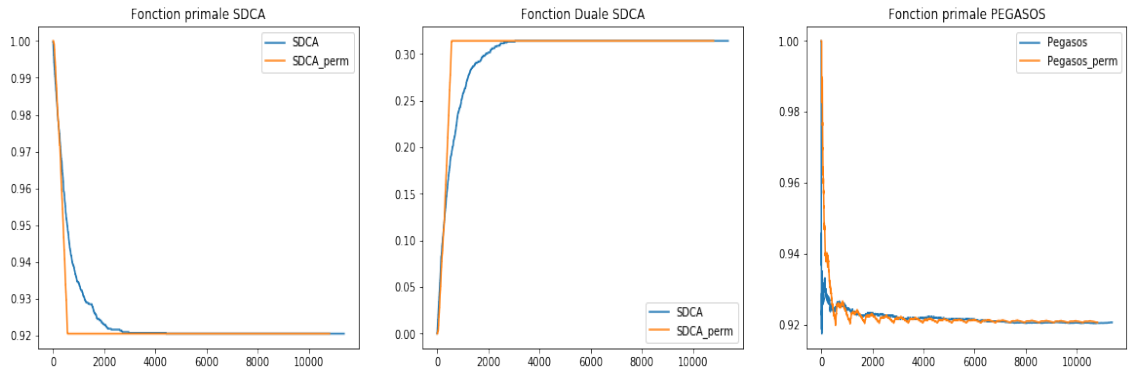
Mise en place de PEGASOSPERM

Analyse

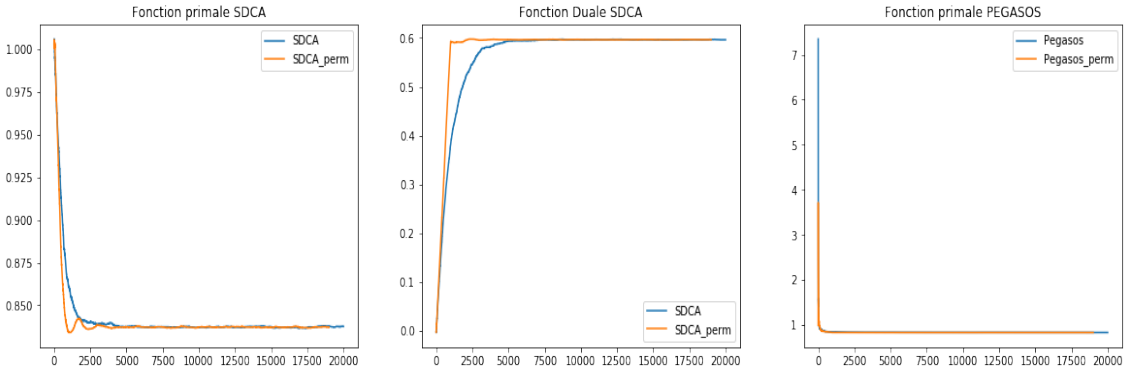
Nous allons à présent analyser nos algorithmes précédents sur deux datasets distincts. Nous allons regarder l'évolution de la convergence de notre minimisation ainsi que la précision finale obtenue dans le problème de classification.

Le premier correspond à un dataset de détection du cancer du sein disponible sur sklearn. Il comporte 569 observations pour 30 variables. Le second dataset est un dataset artificiel effectué à partir de réalisations bruitées d'une loi normale. Il comporte 1000 observations pour 50 variables.

Nous allons dans un premier temps évaluer pour chacun des algorithmes (SDCA et PEGASOS) la différence de convergence pour les deux variantes (avec ou sans permutation) pour les données du cancer du sein :



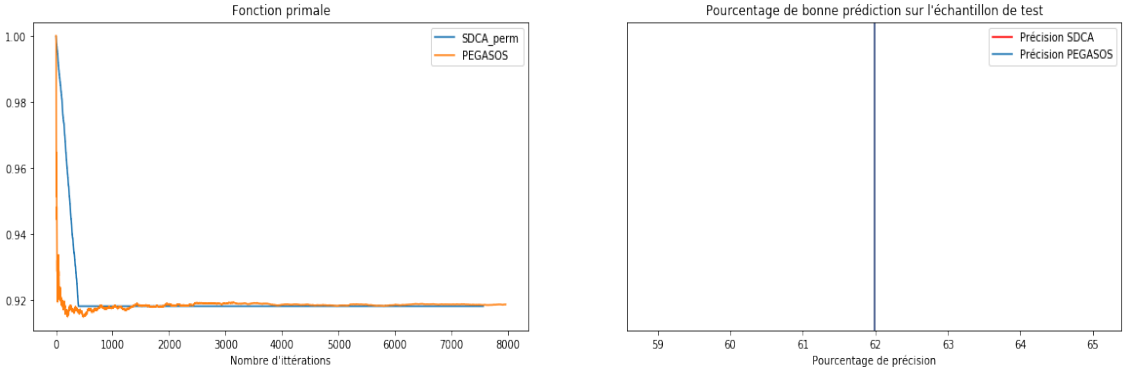
Nous effectuons la même étude concernant cette fois les données simulées :



En terme de convergence la variante "permutation" semble mieux "performer" sur les deux datasets pour l'algorithme SDCA, en revanche l'efficacité de la version "permutation" de PEGASOS a des performances plus discutable sur le premier dataset. Pour la suite nous garderons alors les algorithmes SDCAperm et pegasos pour notre étude.

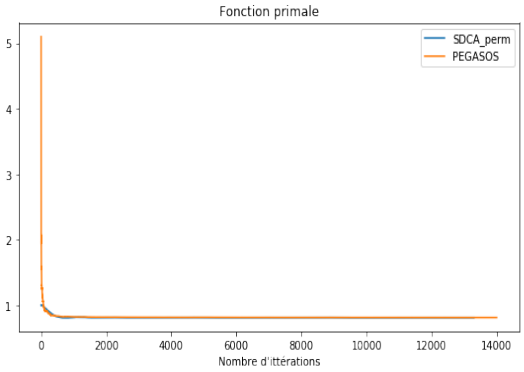
Jusqu'ici,nous avons considéré l'intégralité de nos données pour minimiser la fonction. Nous allons ici utiliser le découpage dataset d'apprentissage/dataset de test pour évaluer la précision des algorithmes également (cf code dans le notebook)

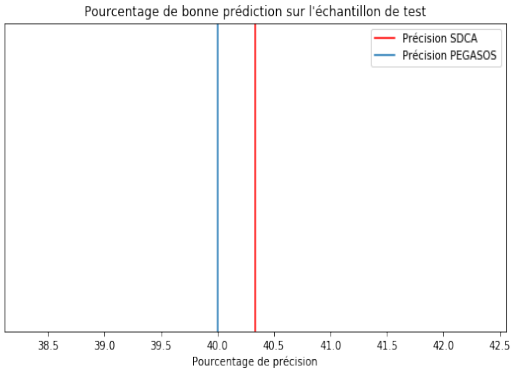
Commençons par le premier dataset portant sur le cancer du sein :



Nos deux algorithmes tendent quasiment vers le même ω ce qui implique un taux de précision similaire (environ 60 %) qui est plutôt faible. En terme de convergence, il semble que l'algorithme PEGASOS décroit plus rapidement vers le $\omega final$ mais mets plus de temps à se stabiliser que l'algorithme SDCAperm. En effet l'algorithme PEGASOS semble se stabiliser au bout de 3000 itérations environ contre 500 pour le SDCA.

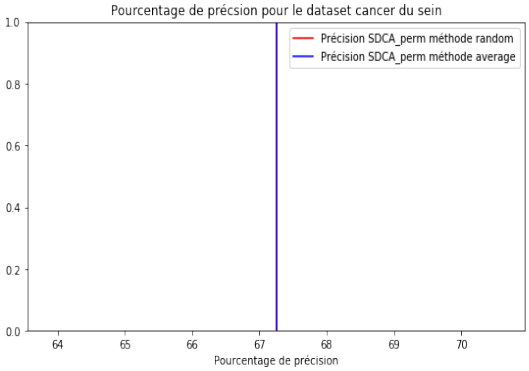
Effectuons à présent la même étude pour le second dataset :

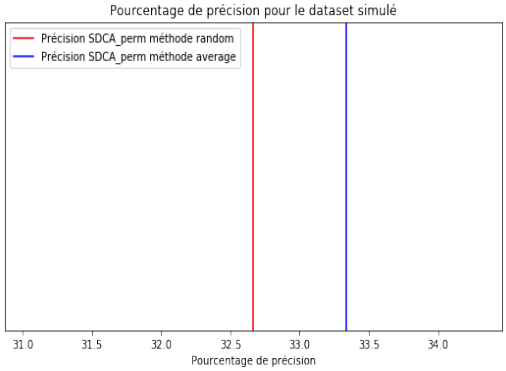




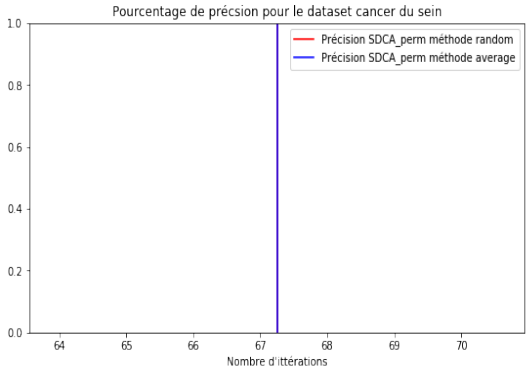
Sur ce dataset, les deux convergences sont très comparables en terme de vitesse et de stabilité. En revanche sur la précision, l'algorithme SDCA semble faire légèrement mieux que son concurrent PEGASOS. Dans les deux cas, puisque nous sommes dans un contexte de classification binaire les résultats ici sont plutôt faibles.

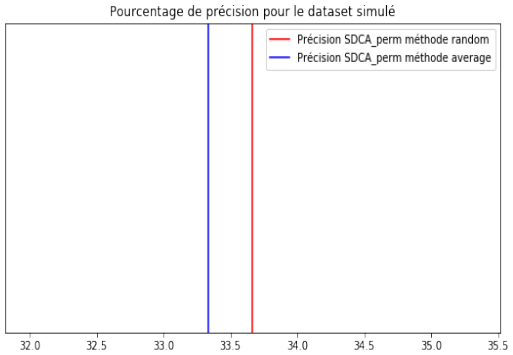
Nous allons à présent observer l'influence de la méthode choisie (random ou average) pour le SDCA sur la précision finale obtenue :





Nous constatons que pour le premier dataset les deux méthodes offrent une précision équivalente. Pour le second dataset un léger écart de précision apparaît. Ceci peut s'expliquer par la dimension plus importante (50 contre 30) du second dataset. La différence d'efficacité des deux méthodes est également liée au paramètre T_0. Ce paramètre permet de fixer à partir de quelle ittération nous allons tirer le $\omega final$ aléatoirement ou à partir de quelle ittération allons nous effectuer la moyenne selon la méthode employé. Etant donné que le comportement de l'algorithme est en phase transitoire sur les premières ittérations, la méthode "average" risque d'être moins performante si T_0 est trop petit. A titre d'illustration nous traçons ci dessous la même étude avec un T_0 dix fois moins important :





On constate alors que la méthode "average" qui avait de meilleures performances que la méthode "random" sur le second dataset réussit moins bien ici. Ceci s'explique par l'influence du régime transitoire qui, si T_0 est trop petit, peut pénaliser cette méthode.

Conclusion

Nous avons dans notre étude implémenté l'algorithme SDCA ainsi qu'une variante avec une stratégie sous Gradient (PEGASOS). Nous avons alors comparer ces algorithmes sous deux variantes, en terme de minimisation du problème primal et de précision de la classification associé sur deux datasets distincts.

Les deux algorithmes ont quasiment les même performances en terme de précision sur nos deux datasets (avec un léger avantage peut être pour le SDCA). Concernant la minimisation de la fonction primale notre étude semble montrer que l'algorithme PEGASOS descend plus vite vers un minimum de la fonction mais mets plus de temps que SDCA à se stabiliser sur les premières itérations.