Projet SOADML

Comparaison des algorithmes SDCA et Pegasos dans le cadre de l'estimation d'un SVM

AUBENEAU Simon

ZIADI Flora

Mastère Spécialisé Data Science

Année 2017-2018

Table des matières

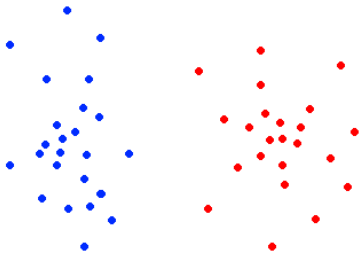
1. Introduction

Dans ce rapport, nous allons décrire et analyser différents types d'algorithmes pour résoudre le problème d'optimisation introduit par les machines à vecteurs de support (SVM).

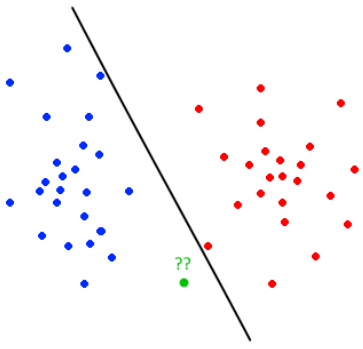
Présentation du SVM :

Les SVM sont des techniques d'apprentissage supervisé permettant de résoudre des problèmes de classification. Leur utilisation peut aussi être étendue aux problèmes de régression.

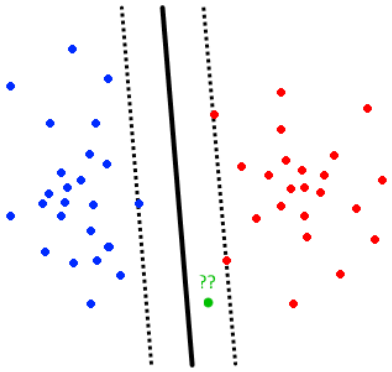
Supposons que nous avons le problème de classification suivant :



Nous cherchons une droite séparatrice, permettant de séparer le plan en deux : nous voulons que tous les points bleus soient situés d'un côté et que tous les points rouges soient situés de l'autre côté.



Cependant, toutes les droites séparant les points ne conviennent pas forcément. Nous souhaitons que la droite apprise ait de bonnes propriétés de généralisation. Dans l'exemple ci-contre, une nouvelle donnée, le point vert, sera attribué à la classe bleue, alors qu'il est plus proche du nuage de points rouge. La droite de séparation n'est pas la plus adaptée.



Nous allons donc chercher la droite entourée par la plus grande marge possible. C'est-à-dire, la droite telle que le point le plus proche parmi les données à classer soit le plus loin possible. Avec cette droite (unique), nous voyons que le point vert sera attribué correctement à la classe rouge.

Les SVM sont la généralisation de ce principe en dimension supérieure ou égale à 2 et dans le cas où tous les points ne sont pas forcément linéairement séparables.

Problème Primal et Dual :

Problème Primal :

Nous considérons le problème d'optimisation générique suivant associé à la minimisation de la perte régularisée de prédicteurs linéaires. Soient les vecteurs $x_1, \dots, x_n \in \mathbb{R}^d$, une suite de fonctions scalaires convexes $\phi_1, \dots, \phi_n$ et le paramètre de régularisation $\lambda > 0$. Le but est de résoudre le problème de minimisation suivant :

$$\min_w P(w) := \left[\frac{1}{n} \sum_{i=1}^{n} \phi_i(w^\top x_i) + \frac{\lambda}{2} \|w\|^2 \right] \quad (1)$$

Problème Dual :

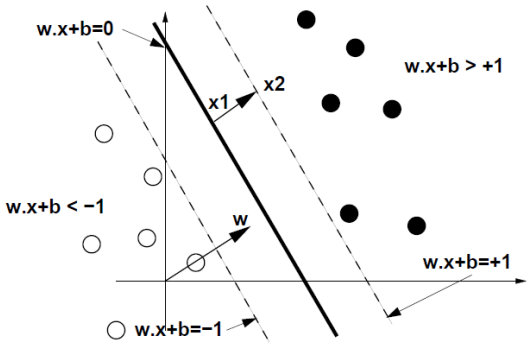
Soit $\forall i,\ \phi_i^* : \mathbb{R} \to \mathbb{R}$ le conjugué convexe de ϕ_i. L'objectif est de maximiser le problème suivant :

$$\max_{\alpha \in \mathbb{R}^n} D(\alpha) := \left[\frac{1}{n} \sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i \right\|^2 \right] \quad (2)$$

L'objectif du problème dual est de résoudre le problème primal en estimant α afin de calculer w.

Dans le cadre du problème SVM, $y_1, \dots, y_n \in \{-1; +1\}$. De plus, si le noyau est linéaire et sans biais, nous pouvons définir ϕ_i par : $\phi_i(u) = \max\{0, 1 - y_i u\}$.

Hyperplan optimal :



Le plan optimal est définit par le couple (w^T, b^*). b^* est défini par l'équation suivante :

$$b^* = -\frac{1}{2}\left(\min_{y_i=1,\alpha_i>0}(\mathbf{w}^T\mathbf{x}_i) + \max_{y_i=-1,\alpha_i>0}(\mathbf{w}^T\mathbf{x}_i)\right)$$

Nous allons étudier des algorithmes basés sur la descente stochastique du sous-gradient (SGD, Pegasos, Pegasos Mini-Batch et Pegasos Kernel) pour résoudre directement le problème primal et les algorithmes SDCA et SDCA Perm pour résoudre le problème dual. L'objectif de ce projet est d'étudier la convergence de ces algorithmes.

Dans un premier temps, nous allons présenter la théorie qui se cache derrière chacun de ces algorithmes. Puis dans un second temps, nous allons analyser la convergence de ces algorithmes sur différents jeux de données et pour différents paramètres de régularisation λ.

2. Présentation théorique des algorithmes

2.1. SGD et méthodes de gradient : Pegasos, Pegasos Min-batch, Pegasos kernel

2.1.1. Pegasos

L'algorithme Pegasos est une méthode de gradient stochastique utilisée pour le calcul des SVM. L'objectif est de minimiser une fonction objective :

$$P(w) = \frac{1}{n}\sum_{i=1}^{n} max\{0,1 - y_i w^T x_i\} + \frac{\lambda}{2}\|w\|^2$$

Pour cela, nous tirons aléatoirement un exemple à chaque pas. Nous allons considérer que cet exemple est représentatif de l'ensemble d'apprentissage, et estimer le gradient à partir de cet exemple particulier.

Le gradient est donc défini selon la formule suivante :$grad_t = \lambda w_t - 1_{[y_{it}\langle w_t | x_{it}\rangle < 1]} y_i x_{it}$

Il est aussi possible d'ajouter une étape de projection sur la boule de rayon $1/\sqrt{\lambda}$. Cette étape de projection permet de s'assurer que la norme de w reste petite à chaque itération. Selon les chercheurs, l'ajout de cette étape ne modifie pas substantiellement les performances de l'algorithme.

2.1.2. Stochastic Gradient Descent (SGD)

Nous avons comparé l'algorithme Pegasos à un autre algorithme de descente de gradient stochastique. Cet autre algorithme utilise une définition légèrement différente de la fonction de perte, et a donc des résultats différents. Cet algorithme a parfois des résultats meilleurs que Pegasos, c'est pourquoi nous l'avons inclus dans nos comparaisons.

$$P(w) = \sum_{i=1}^{n} max\{0,1 - y_i w^T x_i\} + \lambda\|w\|^2$$

2.1.3. Pegasos Mini-Batch

Il est possible d'améliorer la stabilité de l'algorithme Pegasos en utilisant la technique du mini-batch. L'idée est de tirer aléatoirement plusieurs exemples à chaque étape au lieu d'un seul. Le gradient est alors estimé à partir d'un échantillon de données. L'estimation du gradient de w est alors plus précise, tout en évitant de parcourir l'ensemble des données à chaque itération.

Pour la suite, nous avons utilisé des échantillons mini-batchs de taille 50.

2.1.4. Pegasos Kernel

L'algorithme Pegasos est compatible avec l'utilisation d'un noyau. L'intérêt des noyaux est la possibilité d'appliquer les SVM à des problèmes non-linéaires. Le noyau transforme les données d'entrées et les plonge dans un espace de plus grande dimension, dans lequel il existe une séparation linéaire des données, permettant d'appliquer l'algorithme Pegasos. Trouver une séparation linéaire dans l'espace en grande dimension permet donc de trouver une séparation non-linéaire dans l'espace d'origine.

Pour appliquer une méthode à noyau, nous devons redéfinir le produit scalaire entre deux vecteurs x et y de l'espace d'origine. Pour cela, on définit la fonction de transformation Φ qui associe à tout vecteur de l'espace d'origine un vecteur dans l'espace en plus grande dimension. On définit ensuite le noyau de la manière suivante : $K(x, y) = \langle \Phi(x) | \Phi(y) \rangle$

Il n'est pas toujours nécessaire d'exprimer explicitement la fonction Φ. On peut en effet se donner une fonction K définie-positive. La fonction Φ est alors implicite.

La fonction de décision pour un noyau devient alors :

$$f(x) = \sum_{i=1}^{n} y_i \, \alpha_i K(x_i, x) + b$$

Nous avons utilisé deux types de noyaux : le noyau linéaire et le noyau gaussien.

Nous avons testé les deux noyaux sur des données linéairement séparables, puis sur le jeu de données paraboliques.

2.2. Stochastic Dual Coordinate Ascent (SDCA)

Une approche alternative au SGD est l'algorithme DCA (Dual Coordinate Ascent) qui permet de résoudre un problème dual de l'équation (1).

Soit $\forall i,\ \phi_i^*: \mathbb{R} \to \mathbb{R}$ le conjugué convexe de ϕ_i.

Le problème dual est le suivant :

$$\max_{\alpha \in \mathbb{R}^n} D(\alpha) \quad \text{where} \quad D(\alpha) = \left[\frac{1}{n} \sum_{i=1}^{n} -\phi_i^*(-\alpha_i) - \frac{\lambda}{2} \left\| \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i \right\|^2 \right] \quad (2)$$

Le problème dual (2) possède pour chaque exemple dans la base d'entrainement, une variable duale différente. Pour chaque itération du DCA, le problème dual est optimisé vis-à-vis d'une variable duale alors que le reste des variables duales est fixe.

Le problème dual permet de résoudre le problème primal. On va estimer α afin de calculer w.

Si on définit donc w de la façon suivante :

$$w(\alpha) = \frac{1}{\lambda n} \sum_{i=1}^{n} \alpha_i x_i$$

Alors on a que $(\alpha^*) = w^*$, où w^* est la solution optimale de (1) et α^* est la solution optimale de (2). Il est aussi connu que $P(w^*) = D(\alpha^*)$, ce qui implique que pour tout w and α $P(w) \geq D(\alpha)$. Alors l'écart de dualité définit comme $P(w(\alpha)) - D(\alpha)$ peut être vu comme la limite supérieure de la sous-optimalité primale $P(w(\alpha)) - P(w^*)$.

Nous nous concentrons ici sur une version stochastique du DCA, notée SDCA, pour laquelle à chaque itération nous choisissons aléatoirement une variable duale sur laquelle est faite l'optimisation.

Dans ce rapport, nous allons étudier deux versions du SDCA :

- Le SDCA classique : A chaque itération, l'algorithme SDCA génère un α_i aléatoirement.
- Le SDCA-Perm : Au lieu de la générer aléatoirement i, l'algorithme SDCA-Perm parcourt les époques (i.e. l'ensemble du jeu de données), et chaque époque utilise une permutation aléatoire des données.

Les pseudo-codes de ces deux algorithmes sont les suivants :

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 $\bar{w}$

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

Let $w^{(0)} = w(\alpha^{(0)})$
Let $t = 0$
Iterate: for epoch $k = 1, 2, \ldots$
  Let $\{i_1, \ldots, i_n\}$ be a random permutation of $\{1, \ldots, n\}$
  Iterate: for $j = 1, 2, \ldots, n$:
    $t \leftarrow t + 1$
    $i = i_j$
    Find $\Delta\alpha_i$ to increase dual (*)
    $\alpha^{(t)} \leftarrow \alpha^{(t-1)} + \Delta\alpha_i e_i$
    $w^{(t)} \leftarrow w^{(t-1)} + (\lambda n)^{-1}\Delta\alpha_i x_i$
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 $\bar{w}$

Pour rappel, nous utilisons la fonction hinge loss pour un SVM:

$$\phi_i(u) = \max\{0, 1 - y_i u\}$$

Pour la hinge loss, dans le cadre du SDCA et SDCA perm, on peut utiliser la formule fermée suivante pour calculer $\Delta\alpha_i$:

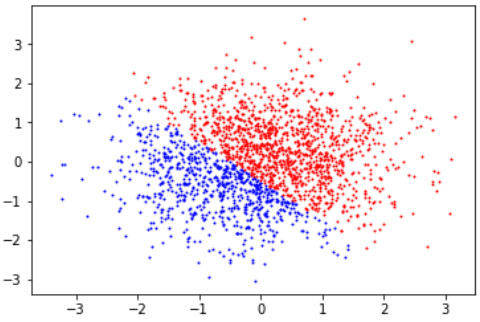
$$\Delta\alpha_i = y_i \max\left(0, \min\left(1, \frac{1 - x_i^\top w^{(t-1)} y_i}{\|x_i\|^2/(\lambda n)} + \alpha_i^{(t-1)} y_i\right)\right) - \alpha_i^{(t-1)}$$

3. Résultats expérimentaux

3.1. Présentation des jeux de données

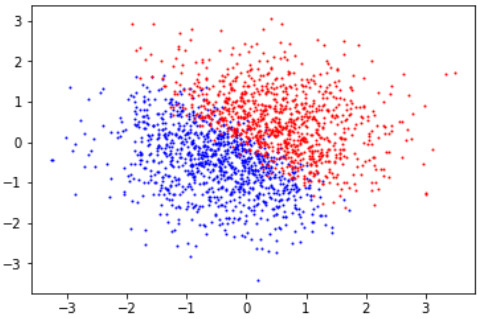
Pour évaluer les algorithmes, nous avons utilisé quatre jeux de données (trois jeux de données simulées et un jeu de données réelles) :

- Jeu 1 : Données linéaires séparables



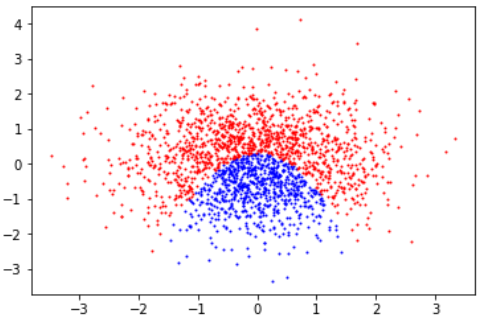
Il s'agit d'un jeu de données en deux dimensions généré à partir d'une loi normale. Les données sont réparties en deux groupes séparables par une droite.

- Jeu 2 : Données linéaires non séparables



Il s'agit d'un jeu de données en deux dimensions généré à partir d'une loi normale. Les données sont réparties en deux groupes non séparables par une droite.

- Jeu 3 : Données paraboliques



Il s'agit d'un jeu de données en deux dimensions généré à partir d'une loi normale. Les données sont réparties en deux groupes séparables par une parabole.

- Jeu 4 : Données réelles « Breast Cancer »

Il s'agit d'un jeu de données de classification binaire en dimension 30, qui porte sur le cancer du sein. Nous avons dû normaliser les données, car les algorithmes ne convergeaient pas sur les données initiales.

3.2. Analyse de la convergence des algorithmes

3.2.1. Analyse en fonction de la structure des données

Dans cette partie, nous avons tracé la fonction de minimisation ainsi que l'erreur de classification en fonction du nombre d'époques (i.e. le nombre de fois que l'algorithme parcourt l'ensemble des données) et du temps pour chaque jeu de données et chaque algorithme à l'exception de l'algorithme Pegasos kernel. En effet, comme les temps de calcul pour l'algorithme Pegasos kernel sont assez longs et qu'il est plus adapté aux données séparées par une parabole, nous lui réservons une étude spécifique à la section 3.2.

La fonction de minimisation correspond au problème primal et est définie de la façon suivante :

$$P(w) = \left[\frac{1}{n}\sum_{i=1}^{n} max\{0,1 - y_i w^T x_i\} + \frac{\lambda}{2}\|w\|^2\right]$$

L'erreur de classification est définie comme le nombre de y_i mal classé divisé par la taille de l'échantillon n.

$$Erreur\ de\ classification = \frac{\sum_{i=1}^{n} 1_{\{y_i \neq \hat{y}_i\}}}{n}$$

Analyse du Jeu 1 et 2 : Données linéaires séparables et non-séparables

Pour la fonction de minimisation, les algorithmes SDCA, SDCA Perm, Pegasos et Pegesos Mini-Batch convergent vers la même valeur. Pegasos Mini-Batch atteint cette valeur en premier, puis SDCA et SDCA Perm atteignent cette valeur quelques itérations plus tard. Pegasos n'est pas très stable et oscille autour de cette valeur. Le SDG atteint un minimum plus élevé que les autres algorithmes, néanmoins il est très stable.

Pour l'erreur de classification, les cinq algorithmes convergent vers la même valeur. De plus, l'erreur de classification atteinte est très bonne : 1,5% pour le jeu 1 et 7% pour le jeu 2. Pour le SDG, la convergence en fonction du nombre d'époques et du temps est très stable. Pour les algorithmes SDCA et SDCA Perm, la convergence est un peu moins stable. Pour les algorithmes Pegasos et Pegasos Mini-Batch les résultats ne sont pas très stables et oscillent au tour du minimum.

Analyse du Jeu 3 : Données paraboliques

Pour la fonction de minimisation, les algorithmes convergent vers une même valeur assez élevée. De plus, l'algorithme Pegasos n'est pas très stable.

Pour l'erreur de classification, les cinq algorithmes convergent vers une erreur de classification très élevée de l'ordre de 30%. Hormis le SGD, les autres algorithmes ne sont pas très stables. Nous étudierons plus en profondeur ce jeu de données à la section 3.2 avec l'algorithme Pegasos à noyau.

Analyse du Jeu 4 : Données réelles « Breast Cancer »

Pour la fonction de minimisation, les algorithmes SDCA, SDCA Perm, Pegasos et Pegesos Mini-Batch convergent vers une valeur assez faible et l'algorithme SGD converge vers une valeur plus élevée.

Pour l'erreur de classification, tous les algorithmes obtiennent d'assez bons résultats. Le SGD obtient une erreur de classification proche de 5 % et les autres algorithmes obtiennent une erreur de classification proche de 8 %.

Figure 1 : Fonction de minimisation en fonction du nombre d'époques et du temps ($\lambda = 10^{-3}$)

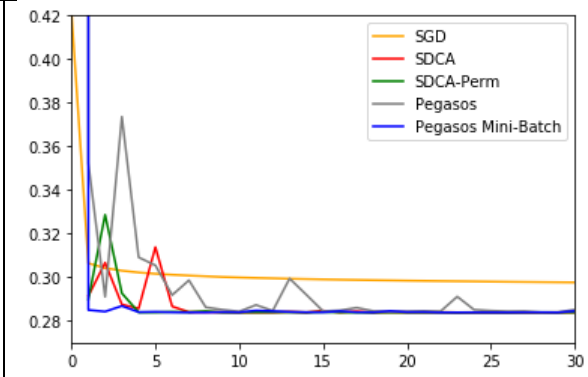
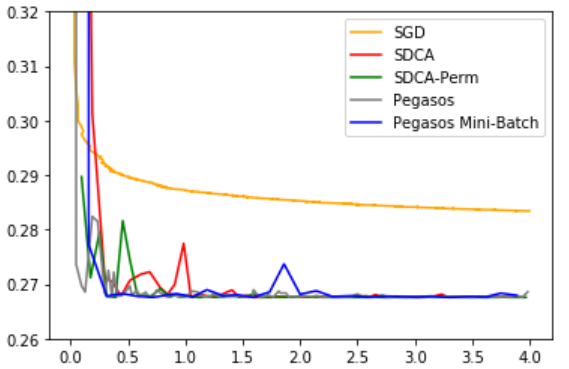
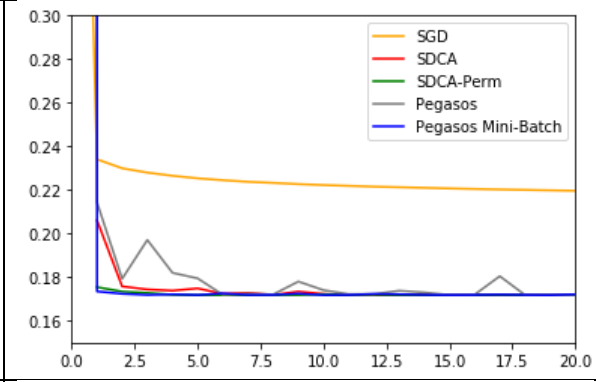
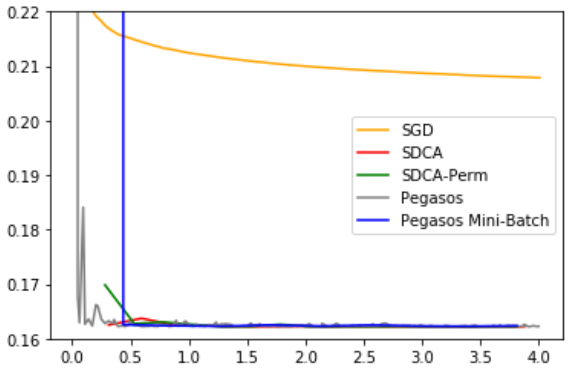
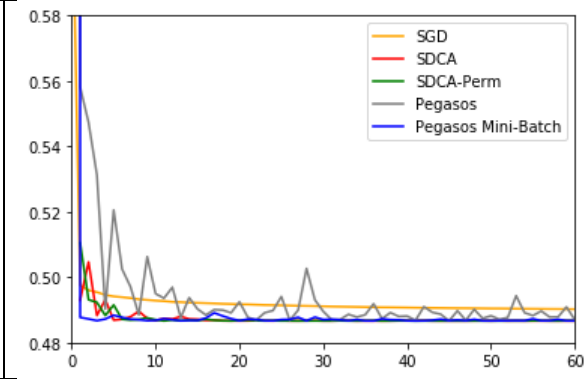
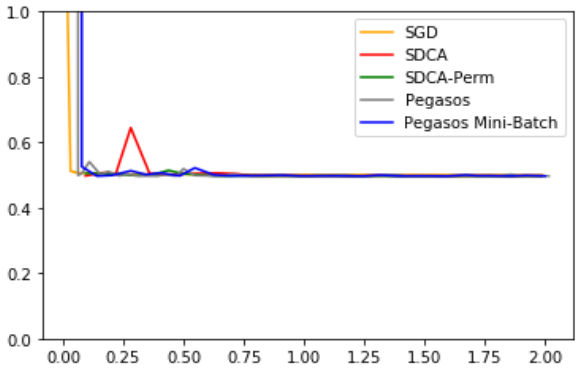
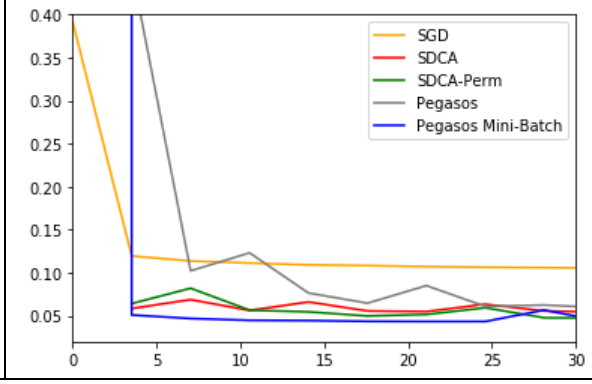
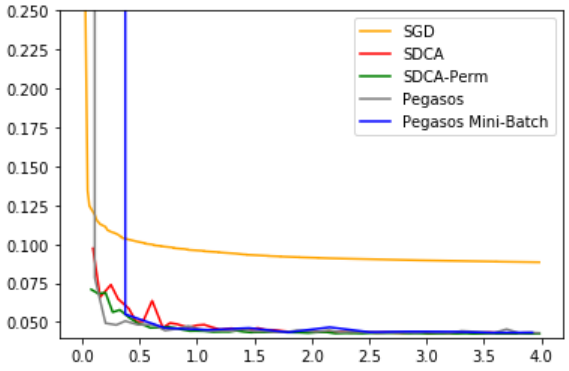
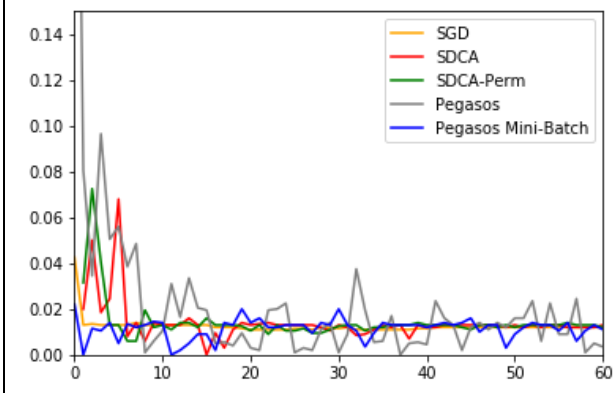
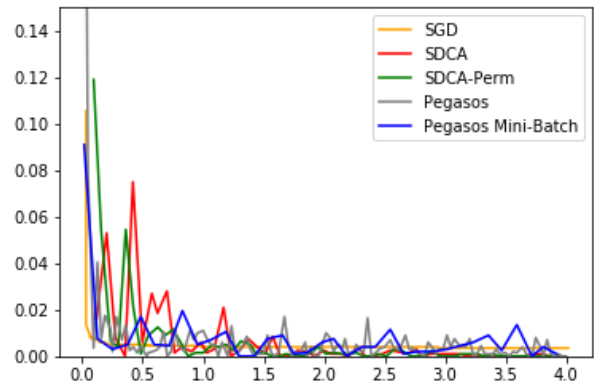
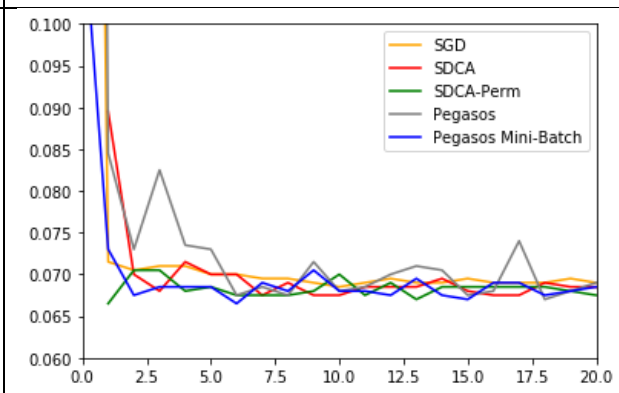
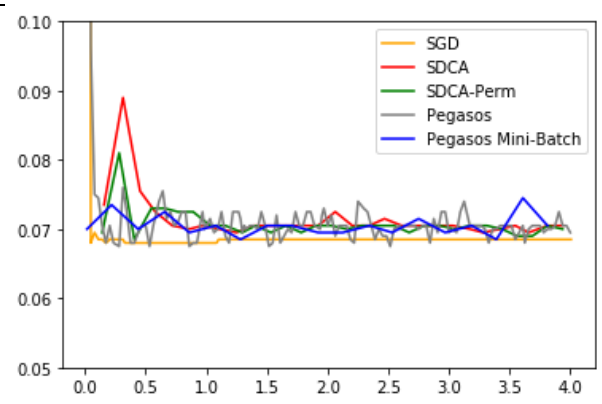
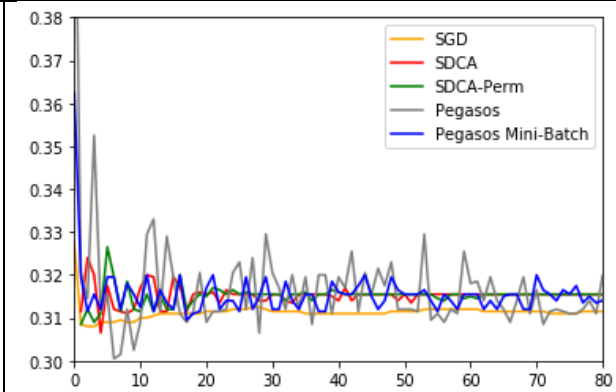
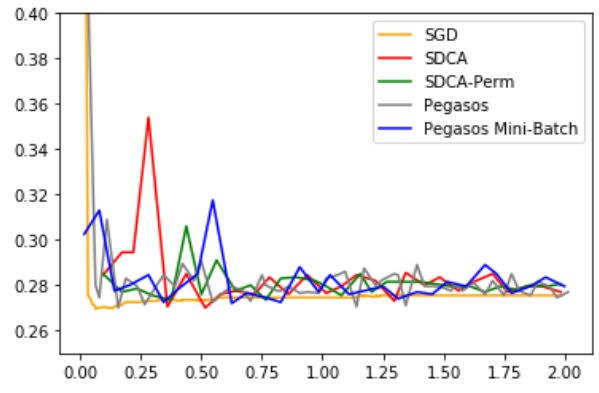
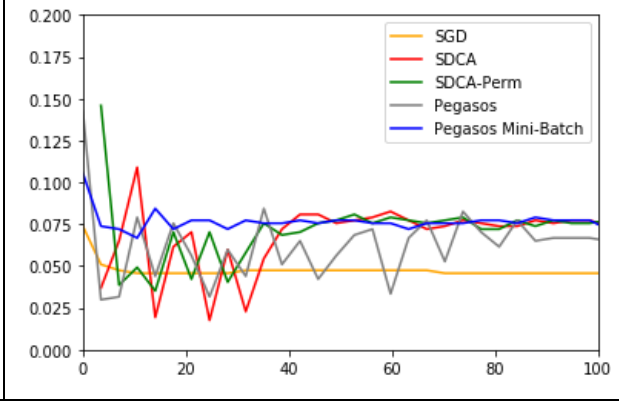
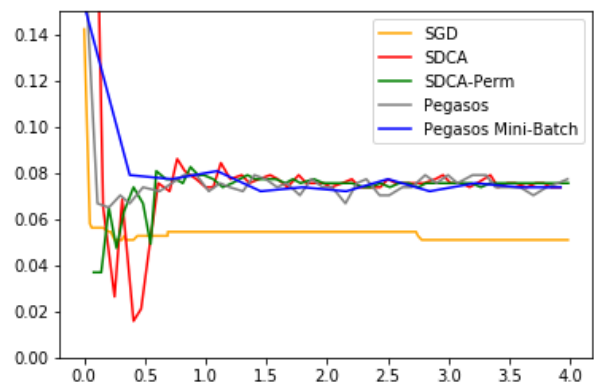
	Fonction de minimisation en fonction du nombre d'époques	Fonction de minimisation en fonction du temps (en seconde)
Jeu 1		
Jeu 2		
Jeu 3		
Jeu 4		

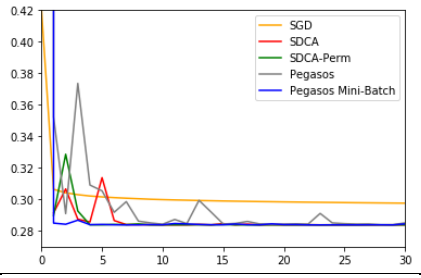
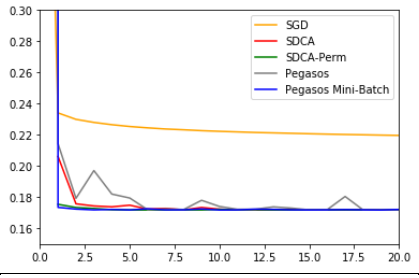
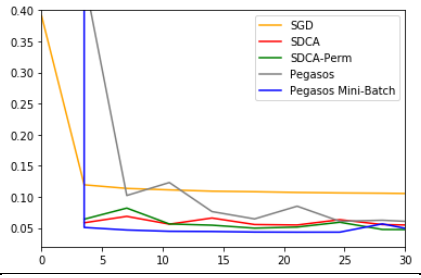
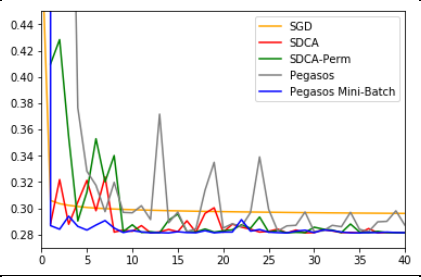
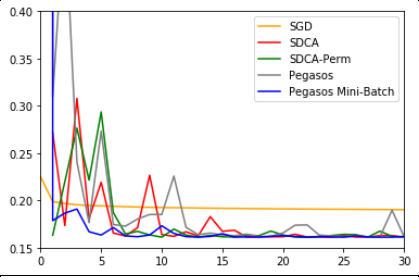
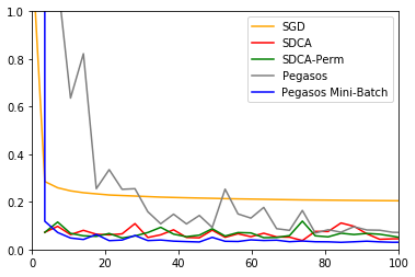
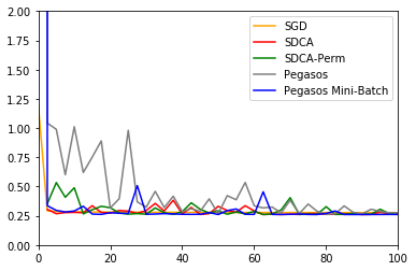
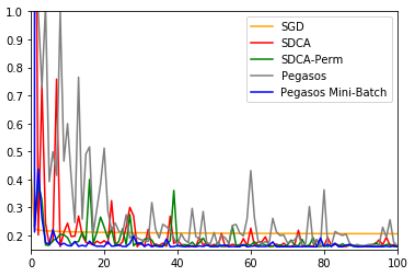
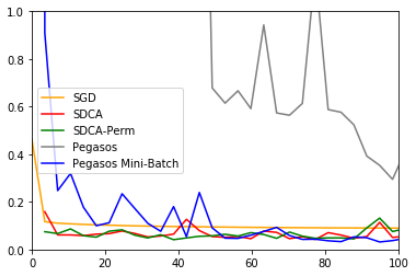
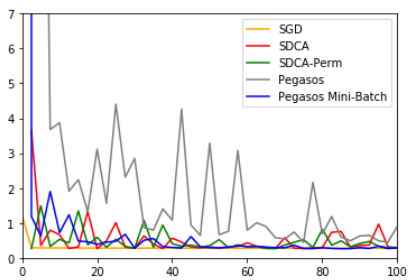
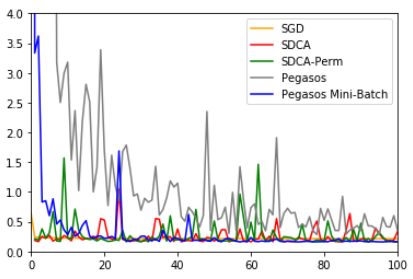
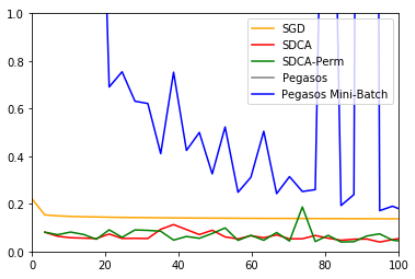
Figure 2 : Erreur de classification en fonction du nombre d'époques et du temps ($\lambda = 10^{-3}$)

	Erreur de classification en fonction du nombre d'époques	Erreur de classification minimisation en fonction du temps (en seconde)
Jeu 1		
Jeu 2		
Jeu 3		
Jeu 4		

3.2.2. Analyse en fonction de la structure des données

Dans cette partie, nous allons étudier le comportement des algorithmes en fonction du paramètre de régularisation λ. Nous avons tracé la fonction de minimisation ainsi que l'erreur de classification en fonction du nombre d'époques et du temps pour des λ différents.

Figure 3 : Fonction de minimisation en fonction du nombre d'époques

λ	Jeu 1 : Données linéaires	Jeu 2 : Données linéaires non séparables	Jeu 4 : Données réels « Breast Cancer »
10^{-3}			
10^{-4}			
10^{-5}			
10^{-6}			

Le valeur de convergence reste identique en fonction des différentes valeurs de λ. Néanmoins, seul le SGD reste réellement stable. Plus λ est faible, moins les algorithmes sont stables. Pour un λ très petit ($\lambda = 10^{-6}$), les algorithmes SDCA et Pegasos Mini-Batch sont un peu plus stables que les deux autres pour les jeux de données 1 et 2 et les algorithmes Pegasos et Pegasos Minibatch ont du mal à converger pour le jeu de données 4.

Figure 4 : Fonction de minimisation en fonction du temps (en seconde)

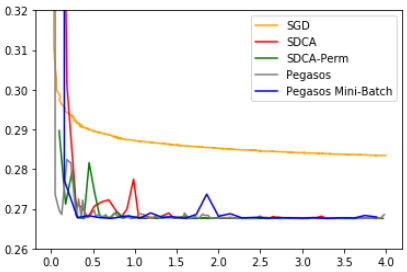
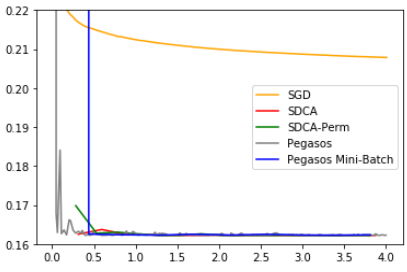
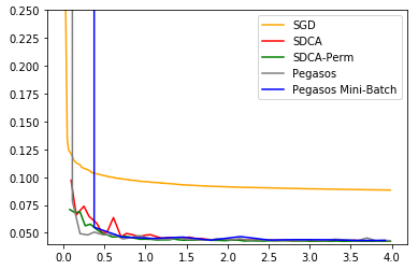
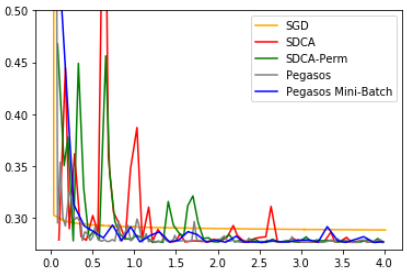
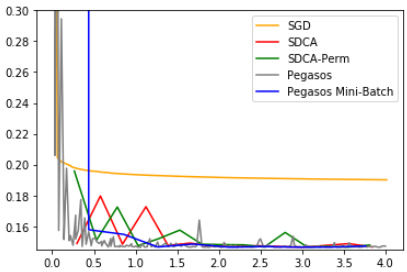
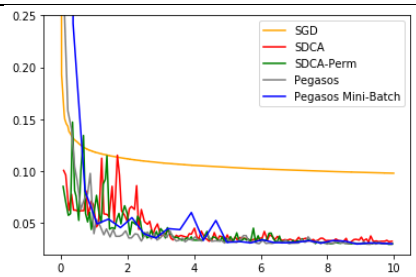
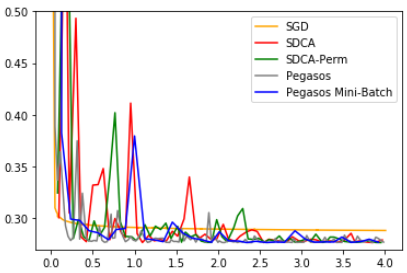
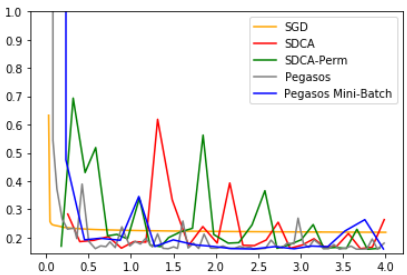
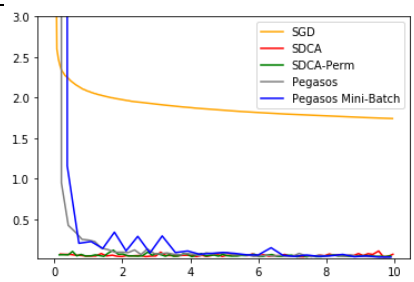
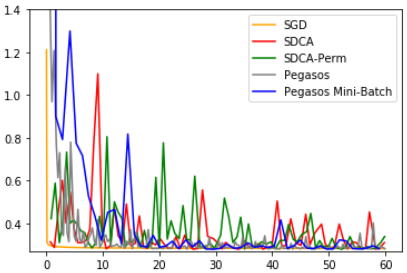
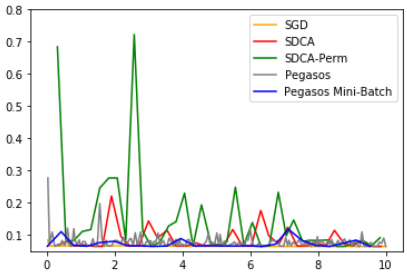
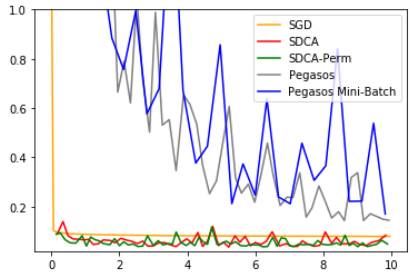
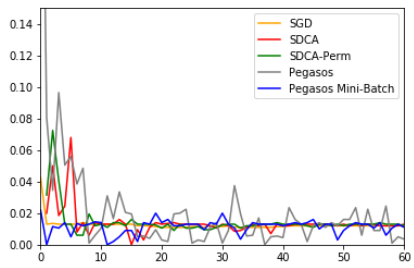
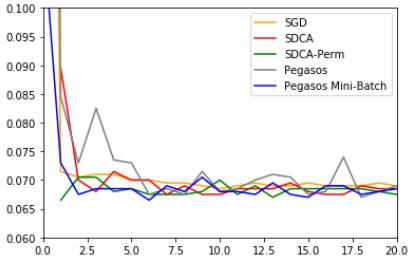
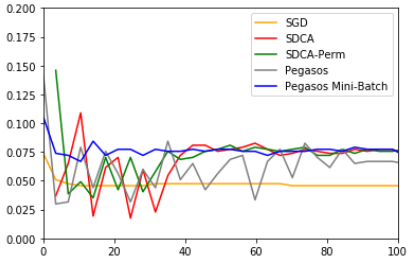
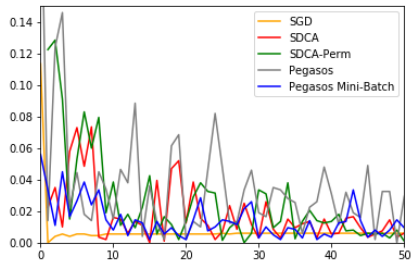
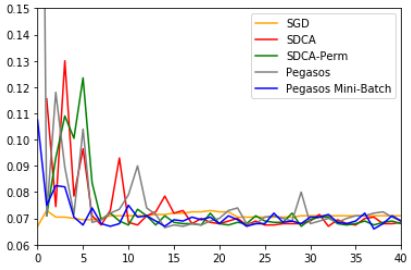
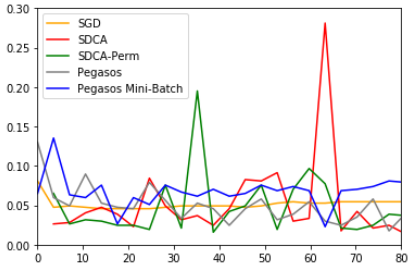
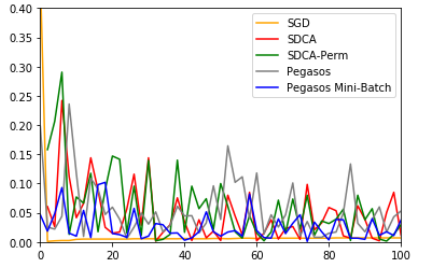
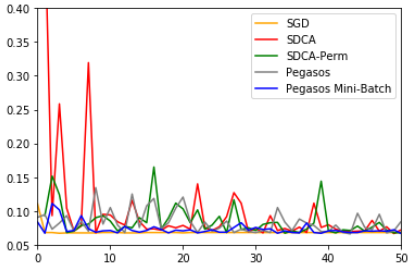
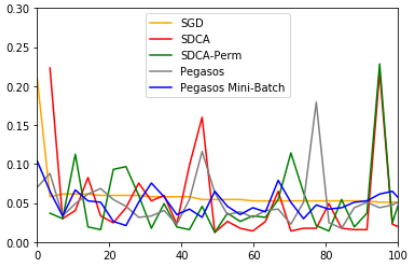
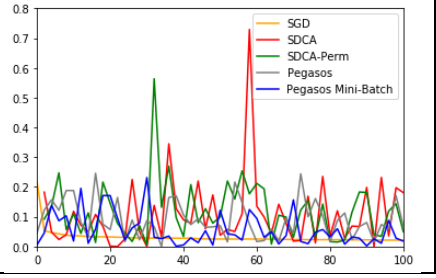
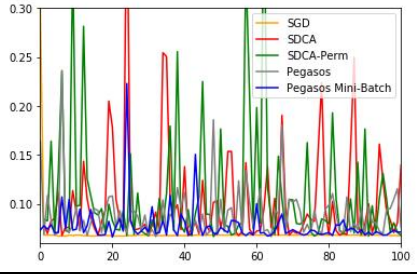
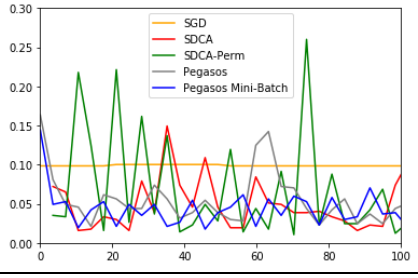
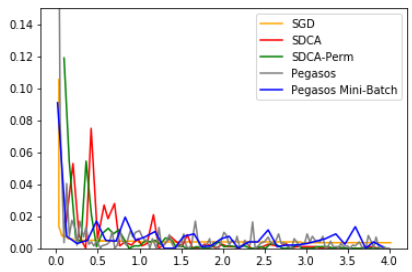
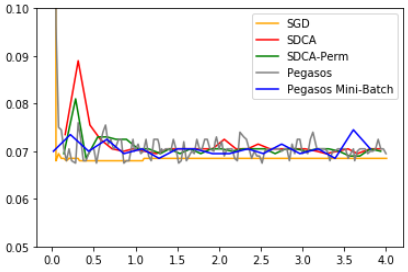
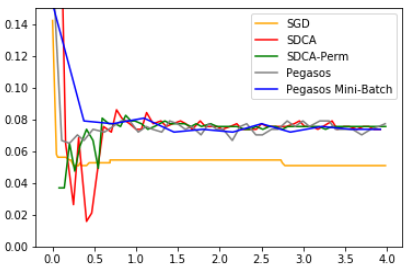
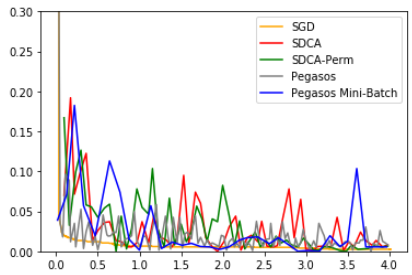
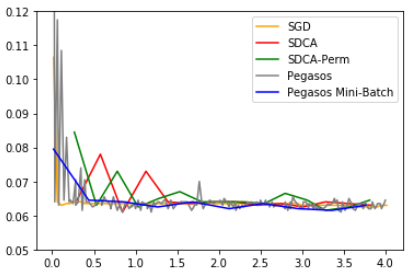
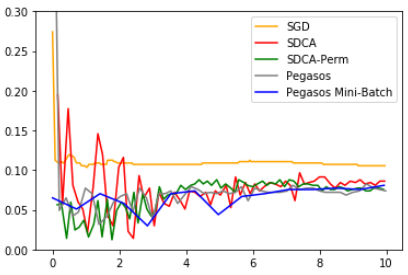
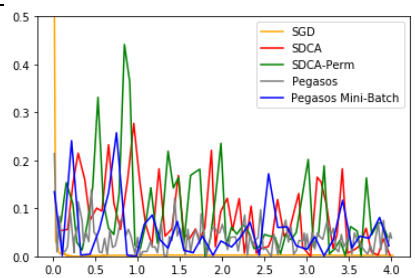
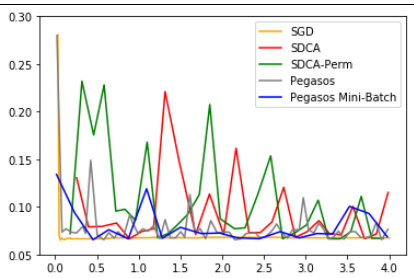
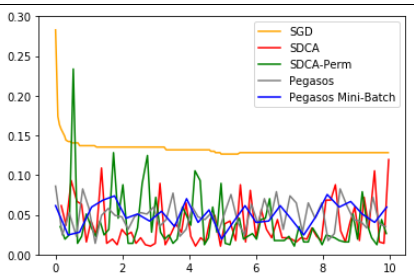
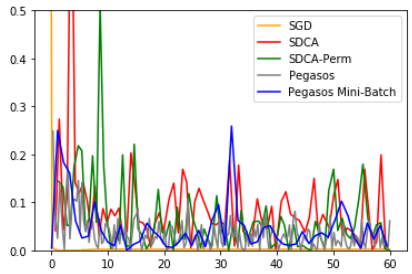
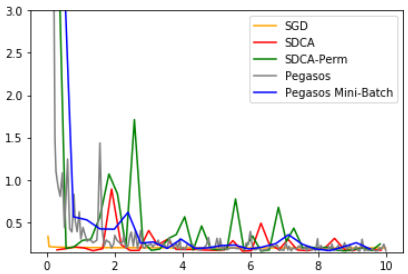
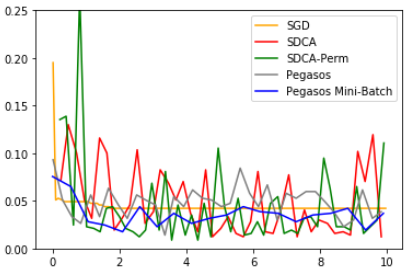
λ	Jeu 1 : Données linéaires	Jeu 2 : Données linéaires non séparables	Jeu 4 : Données réels « Breast Cancer »
10^{-3}			
10^{-4}			
10^{-5}			
10^{-6}			

Figure 5 : Erreur de classification en fonction du nombre d'époques

λ	Jeu 1 : Données linéaires	Jeu 2 : Données linéaires non séparables	Jeu 4 : Données réels « Breast Cancer »
10^{-3}			
10^{-4}			
10^{-5}			
10^{-6}			

Plus λ diminue, plus l'erreur de classification augmente et moins les résultats sont stables.
L'erreur de classification reste constante et stable uniquement pour le SGD.

Figure 6 : Erreur de classification en fonction du temps (en seconde)

λ	Jeu 1 : Données linéaires	Jeu 2 : Données linéaires non séparables	Jeu 4 : Données réelles « Breast Cancer »
10^{-3}			
10^{-4}			
10^{-5}			
10^{-6}			

3.3. Etude spécifique sur l'utilisation d'un noyau

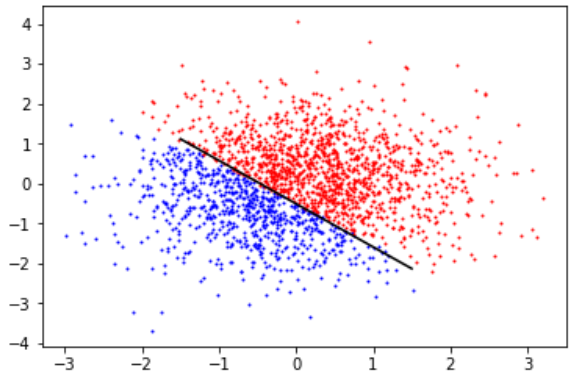
Pour l'algorithme Pegasos-Kernel, nous avons utilisé deux types de noyaux : le noyau linéaire et le noyau gaussien, que nous avons testé sur des données linéairement séparables, puis sur le jeu de données paraboliques.

3.3.1. Jeu de données linéairement séparables

Noyau linéaire :

Sans surprise, le noyau linéaire fonctionne bien pour le jeu de données linéairement séparables. Après 4 secondes de convergence, l'erreur de classification est de 1,4%.

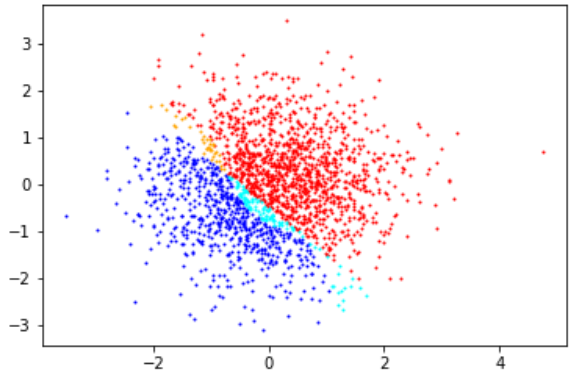
Sur l'image ci-dessous, les deux classes de données sont en rouge et en bleu. La séparation linéaire apprise par l'algorithme est représentée par la ligne noire. On peut vérifier visuellement que la ligne de séparation discrimine les données de manière optimale.



Noyau gaussien :

Le noyau gaussien fonctionne correctement, mais moins bien que le noyau linéaire. Après 4 secondes de convergence, l'erreur de classification est de 7,5%.

Sur l'image ci-dessous, les deux classes de données sont en rouge et en bleu, pour les données bien classées, et en orange et en bleu cyan pour les données mal classées. Cela signifie que l'algorithme attribue à tort les données bleu cyan à la classe rouge et les données oranges à la classe bleue. La séparation est plus complexe que dans le cas linéaire.

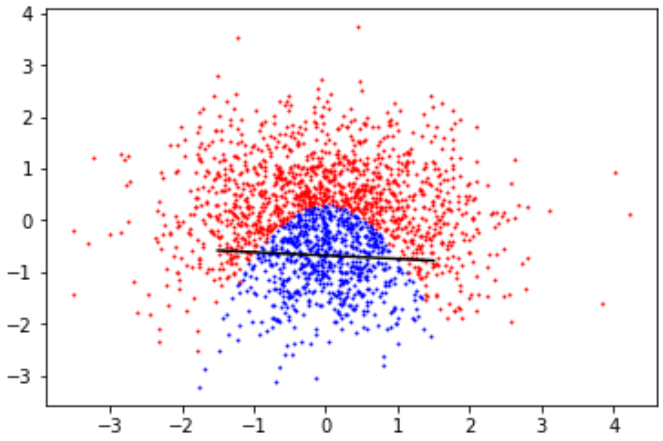


3.3.2. Jeu de données séparables par une hyperbole

Noyau linéaire :

Après 4 secondes de convergence, l'algorithme obtient une erreur de classification de 23,4 %. Les performances du noyau linéaire ont donc chuté lorsque les données ne sont pas linéairement séparables.

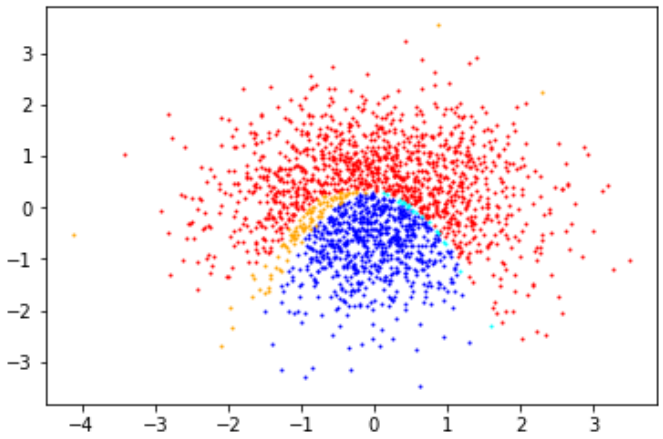
Sur le graphique ci-dessous, nous visualisons l'hyperplan trouvé par l'algorithme.



Noyau gaussien :

Après 4 secondes de convergence, le noyau gaussien obtient une erreur de classification de 6,0 % sur le jeu de données paraboliques. Contrairement au noyau linéaire, le noyau gaussien n'a pas de problème pour classer ce jeu de données. L'erreur est même légèrement plus faible sur le jeu de données parabolique que sur le jeu linéaire.

Ci-dessous, nous représentons les points mal classés en orange et en bleu cyan. On peut voir que la séparation trouvée par le noyau gaussien suit à peu près la forme d'une parabole.



Si l'utilisation d'un noyau linéaire est très efficace dans le cas de données linéairement séparables, l'utilisation d'un noyau gaussien est plus polyvalente et permettra à l'algorithme de trouver des séparations plus intéressantes. Il faut noter que l'implémentation de l'algorithme Pegasos avec des kernels est plus lente que les implémentations des algorithmes Pegasos et Pegasos-Batch. Pour le choix de la méthode, il faudra donc choisir le bon compromis entre temps de calcul et polyvalence de la méthode utilisée.

4. Conclusion

Nous avons étudié la convergence des 6 algorithmes suivants : SGD, Pegasos, Pegasos Mini-Batch, Pegasos Kernel, SDCA et SDCA Perm.

Nous avons analysé la convergence de ces algorithmes sur différents jeux de données et pour différents paramètres de régularisation λ.

Pour le jeu de données 1 (Données linéaires séparables) et le jeu de données 2 (Données linéaires non séparables), tous les algorithmes obtiennent de bons résultats avec une erreur de classification assez faible.

Pour le jeu de données 4 (Données réelles « Breast Cancer »), le SGD obtient de meilleurs résultats que les autres algorithmes. Néanmoins, l'erreur de classification des autres algorithmes reste correcte.

Pour les trois jeux de données cités ci-dessus, les algorithmes sont très sensibles avec λ très petite. L'erreur de classification reste constante et stable uniquement pour le SGD.

Pour le jeu de données 3 (Données paraboliques), seul l'algorithme Pegasos avec un noyau gaussien obtient de bons résultats.

Le SGD (Stochastic Gradient Descent) a obtenu des résultats beaucoup plus stables que les autres algorithmes. A l'exception des données paraboliques, nous préconisions d'utiliser le SGD pour résoudre le problème d'optimisation introduit par un SVM.

Dans le cadre de données paraboliques, l'algorithme Pegasos avec l'utilisation d'un noyau gaussien est plus polyvalent et permet de trouver des séparations plus intéressantes. Toutefois, il est important de noter que les temps de traitement de cet algorithme sont très longs.