ANNÉE 2014

Université de Nationale de Hanoi - Agence universitaire de la Francophonie
Institut de la Francophonie pour l'Informatique

Algorithme parallèle de Descente de gradient stochastique multi-classes pour la classification d'images

Réalisé par
Quoc-Khai NGUYEN
Promotion 17 de l'IFI

Sous la direction de
Thanh-Nghi DO
Nguyen-Khang PHAM
Professeurs de l'Université de Cantho

Stage en Master 2 réalisé à la laboratoire de traitement intelligent des informations de la faculté des technologies de l'information et de la communication, Université de Cantho

Université de Nationale de Hanoi - Agence universitaire de la Francophonie

Institut de la Francophonie pour l'Informatique

Stage en Master 2 réalisé à la laboratoire de traitement intelligent des informations de la faculté des technologies de l'information et de la communication, Université de Cantho.

Réalisé par
Quoc-Khai NGUYEN
Promotion 17 de l'IFI

Sous la direction de
Thanh-Nghi DO
Nguyen-Khang PHAM
Professeurs de l'Université de Cantho

Try not to become a man of success, but rather to become a man of value ....

Remerciement

Ces travaux ont été effectués en collaboration entre l'Université nationale de Hanoi - Agence universitaire de la Francophonie, au sein de l'Institut de la Francophonie pour l'informatique et l'université de Cantho, au sein de la faculté des technologies de l'information. L'IDPL est une environnement qui permet de faire de la recherche dans de bonnes conditions.

Je tiens à exprimer en premier lieu toute ma gratitude envers mes deux encadreurs M.Thanh-Nghi DO et M.Nguyen-Khang PHAM. J'ai grandement apprécié leur encadrement, que ce soit au niveau de la rédaction - où leurs conseils et leurs rigueurs m'ont aidé à aller dans la vrais direction et à améliorer le résultat de mon travail.

J'aimerais remercier M.Tuong-Vinh HO et M.Xuan-Hiep HUYNH par leur accord et leur préparation des conditions pour mon stage. Je souhaiterais remercier tous mes professeurs, surtout, les professeurs qui m'ont enseigné les cours à l'IFI concernant mon stage : M.Rémy Mullot, M.Tuong-Vinh HO, Mme.Thi-Oanh NGUYEN, M.Benoît Frénay.

Table des matières

Résumé

La classification d'images consiste à étiqueter automatiquement des images en catégories prédéfinies. Son application se compose plusieurs domaines importants.

Ce projet consiste à étudier les problèmes concernant la classification d'images et à développer un algorithme parallèle multi-classes basé sur la descente de gradient stochastique. Dans un premier temps, on extrait des données visuelles dans des images. Nous avons d'abord étudié la représentation des images par des vecteurs caractéristiques (SIFT)[1]. L'étape suivante consiste à construire un vocabulaire visuel en appliquant l'algorithme de clustering, k-moyenne sur un ensemble de vecteurs caractéristiques. Un cluster correspond à un mot visuel. Enfin, une image s'est représentée par un histogramme des mots visuels. Cette approche s'inspire au modèle sac-de-mots largement utilisé dans l'analyse des données textuelles. Dans un second temps, nous nous concentrons sur le problème d'apprentissage automatique basé sur la descente de gradient stochastique. Se basant sur l'implémentation SGD binaire Pegasos dans [2], nous avons développé l'algorithme MC-SGD pour la classification multi-classes. Afin d'améliorer la vitesse de l'algorithme sur des machines multi-cœurs, nous avons aussi parallélisé cet algorithme en utilisant l'OpenMP.

Nous constatons que les résultats de notre algorithme sont similaires à ceux de la LibSVM. De plus, notre algorithme est beaucoup plus rapide que la LibSVM, surtout pour les données complexes. Donc, notre méthode s'adapte bien pour la classification d'images où les données sont grandes.

Abstract

Image classification is to automatically tag images in predefined categories. Its application is made several important domains.

This project is to study the problems that concerns the image classification and to develop a parallel multi-class algorithm based on the stochastic gradient descent. Initially, visual data is extracted from images. We first studied the representation of images by feature vectors (SIFT) [1]. The next step is to construct a visual vocabulary by applying of the clustering algorithm, k-mean on the set of feature vectors. A cluster corresponds to a visual word. Finally, an image is represented by a histogram of visual words. This approach is based on model bag of visual words widely used in the analysis of textual data. In a second step, we focus on the machine learning problem based on the stochastic gradient descent. Based on the implementation binary SGD in Pegasos [2], we have developed the MC-SGD algorithm for multi-class classification. To improve the speed of the algorithm on multi-core machine, we also parallelize the algorithm using OpenMP.

We note the results of our algorithm are similar to those of LibSVM. In addition, our algorithm is much faster than LibSVM, especially for complex data. So our method is well suited for image classification where the data are large.

Introduction générale

La classification ou la catégorie des images est importante pour accéder à l'information visuelle au niveau d'objets, qui consiste à étiqueter automatiquement des images en catégories prédéfinies. Ces méthodes sont largement utilisées dans les domaines importants : la reconnaissance des scènes naturelles, la reconnaissance des chiffres sur des chèques, la reconnaissance des codes postaux pour la classification automatique des courriers, la reconnaissance des visages pour l'authentification, etc.

Actuellement, le premier stage de la classification d'image est réalisé par extraire des données visuelles dans des images. Le deuxième stage de la classification d'image est qu'on applique une méthode de classification sur des données visuelles pour la classification. Ce processus trouve le problème de données complexes. Particulièrement, la sortie du premier stage comprend la large dimension et le nombre d'exemples est beaucoup.

Pour les bases de données complexes, la vitesse d'apprentissage des méthodes de classification actuelle est base. Faire face de ce problème, on a deux options pour l'optimisation. L'une se concentre sur la diminution de la dimension des données, l'autre se concentre sur l'amélioration de la vitesse du stage d'apprentissage. Ce stage est mise le point sur le développement d'un algorithme de classification qui peut prendre des données entrées très complexes. C'est à dire on a choisi la deuxième option, améliorer la vitesse de l'étape d'apprentissage.

Pour le stage de classification, il existent plusieurs méthodes tels que : réseau de neurones, quantification vectorielle, arbre de décision, machine des vecteurs de support, etc. Parmi les différents méthodes, la méthode SVM est souvent choisie. Pendant ce stage, nous avons choisi la méthode Descente de Gradient Stochastique (SGD) pour remplacer la méthode SVM en raison de sa simplification et sa efficacité. Au lieu de résoudre le problème de programme quadratique comme la méthode SVM, la méthode SGD apprend par

la descente de gradient stochastique. Nous trouvons que la méthode SGD est beaucoup plus rapide que la méthode SVM. Basé sur le SGD binaire, dans ce projet, nous avons développé un algorithme MC-SGD pour la classification multi-classes et le parallélisé. Nous avons aussi développé l'application MC-SGD-Toy pour mieux comprendre ce que le MC-SGD fait à l'interface.

Description par chapitre

Avant de parler de notre travail, dans premier temps, nous allons présenter la théorie de base des méthodes utilisées. Tout d'abord, nous allons présenter la méthode SIFT et le modèle Sac de Mots dans le chapitre 1. Ensuite, nous allons présenter l'étape d'apprentissage automatique qui se compose la méthode SVM et la méthode SGD dans le chapitre 2. Dans ce chapitre, nous parlerons aussi des façons pour résoudre le problème de multi-classes se basant sur un classificateur de 2 classes. Dans le second temps, nous présenterons notre implémentation dans le chapitre 3. Pour le chapitre 4, nous présenterons le résultat obtenu et l'analyserons en détaillé. A la fin de ce rapport, nous terminerons avec la conclusion et perspective.

Chapitre 1

Extraction des caractéristiques visuelles

1.1 Introduction

Définition 1. ***Caractéristique visuelle***

Une caractéristique d'une image est définie comme une abstraction des informations visuelles de l'image qui sont sélectionnée pour des tâches de calcul reliées à une certaine application (par exemple : classification d'images, recherche d'images).

Les caractéristiques sont extraites soit globalement sur une image entière, soit sur une petite groupe de pixel (une région) d'une image. Le résultat d'une étape d'extraction de caractéristiques (globales ou locales) est appelé *descripteur de caractéristiques.*

Définition 2. ***Descripteur de caractéristiques***

Nous appelons la description mathématique d'une image ou une région locale de l'image après une étape d'extraction de caractéristiques sont ***descripteur de caractéristiques***

Les descripteurs se présentent normalement sous forme d'une vecteur dans un espace vectoriel, $\mathbb{R}^D$, appelé *l'espace de caractéristiques.*

Dans le cas d'une extraction globale, on récupère une seule descripteur par image tandis qu'une description locale permet d'obtenir d'un ensemble de descripteurs locaux pour une image.

Jusqu'à maintenant, les recherches se basent plusieurs types de caractéristiques pour la classification ou la reconnaissance d'images. Nous pouvons lister quelques types de caractéristiques les plus couramment utilisées pour calculer des descripteur tels que : *la couleur, la texture, la forme, les points d'intérêt et les relations spatiales* qui sont décrit dans [4].

1.2 Description locale des images

Afin d'obtenir des descripteurs locaux à partir une image, on commence par extraire des régions. La façon la plus simple est d'utiliser une *partition* qui découpe l'image en rectangles ou en cercles de même taille. Une telle partition simple ne génère pas de région perceptuelle-ment significatives mais c'est une manière simple d'obtenir des caractéristiques globales de l'image avec une résolution plus fine. Dans la lecture, nous trouvons que les deux approches les plus utilisées pour localiser les régions d'intérêt dans l'image : l'une fournit des régions qui se chevauchent (détection des points d'intérêt) et l'autre segmente l'image en région sans intersection (segmentation d'image)[4]. La première approche est efficace pour la classification d'images, donc, dans cette section nous décrivons la première approche.

Détection des points d'intérêt

Les points d'intérêt sont traditionnellement utilisés pour la *stéréo vision* mais sont utilisés aussi dans la classification d'images. Ils sont déterminés de manière telle qu'un point trouvé dans une image sera aussi trouvé dans une autre image qui diffère légèrement de la première. La signification de tels points spéciaux est due à leur représentation compacte des régions importantes de l'image qui conduit à une indexation efficace, et à leur pouvoir discriminant surtout dans la recherche d'objets.

Un des premiers travaux sur ce sujet [5] utilise un détecteur de Harris [6] pour localiser des points d'intérêt invariants à la rotation. Dans [7], les auteurs montrent que les descripteurs ne peuvent pas être invariants au changement d'échelle si les points d'intérêt extraits ne sont pas invariants eux même au changement d'échelle. Par conséquent, plusieurs détecteurs ont été proposé pour obtenir l'invariance au changement d'échelle des points d'intérêt [8, 1, 10, 11]. La sélection automatique de l'échelle est effectuer en choisissant les *extrema* d'une fonction de l'échelle (par exemple : laplacien normalisé, différence de gaussiennes).

Caractérisation des points d'intérêt

Après avoir détecté des points d'intérêt, pour les utiliser, il faut caractériser la région autour de ces points. La caractérisation d'un point d'intérêt est calculée, à une échelle choisie, sur la région autour de ce point. Différents descripteurs ont été proposés dans la littérature : *Shape context* [12], *Scale Invariant Feature Transform (SIFT)* [11], *PCA-SIFT* [13], *Gradient Location and Orientation Histogram (GLOH)* [14]. Parmi des descripteurs listés, le descripteur SIFT est le plus utilisé. Dans ce travail, nous concentrerons au descripteur SIFT, donc, nous décrivons ce descripteur et cette méthode dans la partie suivante.

1.3 Méthode SIFT (Scale-invariant feature transform)

1.3.1 Description de la méthode SIFT

Dans la lecture, nous trouvons qu'on traduit en français "transformation de caractéristiques visuelles invariante à l'échelle". SIFT est une méthode utilisée dans le domaine de la vision par ordinateur pour détecter et identifier les éléments similaires entre différentes images numériques (éléments de paysages, objets, personnes, etc.). Cette méthode a été développé en 1999 par le chercheur David Lowe [1].

L'étape fondamentale de la méthode SIFT consiste à calculer les *descripteurs SIFT* des images à étudier. Il s'agit d'informations numériques dérivées de l'analyse locale d'une image et qui caractérisent le contenu visuel de cette image de la façon la plus indépendante possible de l'échelle, du cadrage, de l'angle d'observation et de l'exposition (luminosité) [1].

La méthode proposée par Lowe comprend deux parties [11] :

1. Un algorithme de détection de caractéristiques et de calcul de descripteurs

2. Un algorithme de mise en correspondance proprement dit

Dans ces deux aspects, le premier est celui qui a le plus assuré la popularité de la méthode [15]. La deuxième permet d'utiliser le résultat de la première partie pour l'usage de la méthode. Dans notre travail, nous utilisons la méthode SIFT comme une étape de base. C'est à dire, nous n'utilisons que

la première partie de la méthode.

La première partie, partie de détection de caractéristiques et de calculer des descripteurs comprend 4 étapes principales [1, 11] :

1. Détection d'extrema dans l'espace des échelles.
2. Localisation précise de points d'intérêt
3. Assignation d'orientation
4. Descripteur de point d'intérêt

Nous décrivons tout de suite pas à pas ces étapes.

1.3.2 Détection d'extrema dans l'espace des échelles

La détection s'effectue dans un espace discret (espace des échelles ou *scale space* en anglais) qui comporte trois dimensions : les coordonnées cartésiennes x et y et le facteur d'échelle σ. Le gradient de facteur d'échelle σ (noté L) est le résultat de la convolution d'une image I par un filtre gaussien G de paramètre d'échelle σ, soit [11] :

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{1.1}$$

Et

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2}$$

Cette convolution a pour effet de lisser l'image originale I de telle sorte que les détails trop petits, c'est-à-dire de rayon inférieur à [1], sont estompés. Par conséquent, la détection des objets de dimension approximativement égale à σ se fait en étudiant l'image appelée différences de gaussiennes (en anglais difference of gaussians, DoG) définie comme suit :

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{1.2}$$

Ou

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$

[1] Ici comme dans la littérature scientifique en général, le facteur d'échelle – paramètre du filtre gaussien σ – est assimilé à une distance en pixels sur l'image, que l'on pourrait appeler rayon associé r. En fait, ils sont proportionnels ($r = \alpha\sigma$), avec un facteur α qui varie généralement entre 3 et 4 selon les auteurs. Il est tout simplement lié au nombre de coefficients au-delà duquel les valeurs de la gaussienne deviennent négligeables.

où k est un paramètre fixe de l'algorithme qui dépend de la finesse de la discrétisation de l'espace des échelles voulue [11].

Pour chaque octave de l'espace des échelle, l'image initiale répétée est fait la convolution avec gaussiennes pour produire l'ensemble des images de l'espace des échelle (image ci-dessous, à gauche). Images gaussiennes adjacentes sont soustraites pour produire les images de différence de gaussienne sur la droite. Après chaque octave, l'image Gaussian est sous-échantillonnée par un facteur de 2, et le processus est répété pour toutes les échelle.

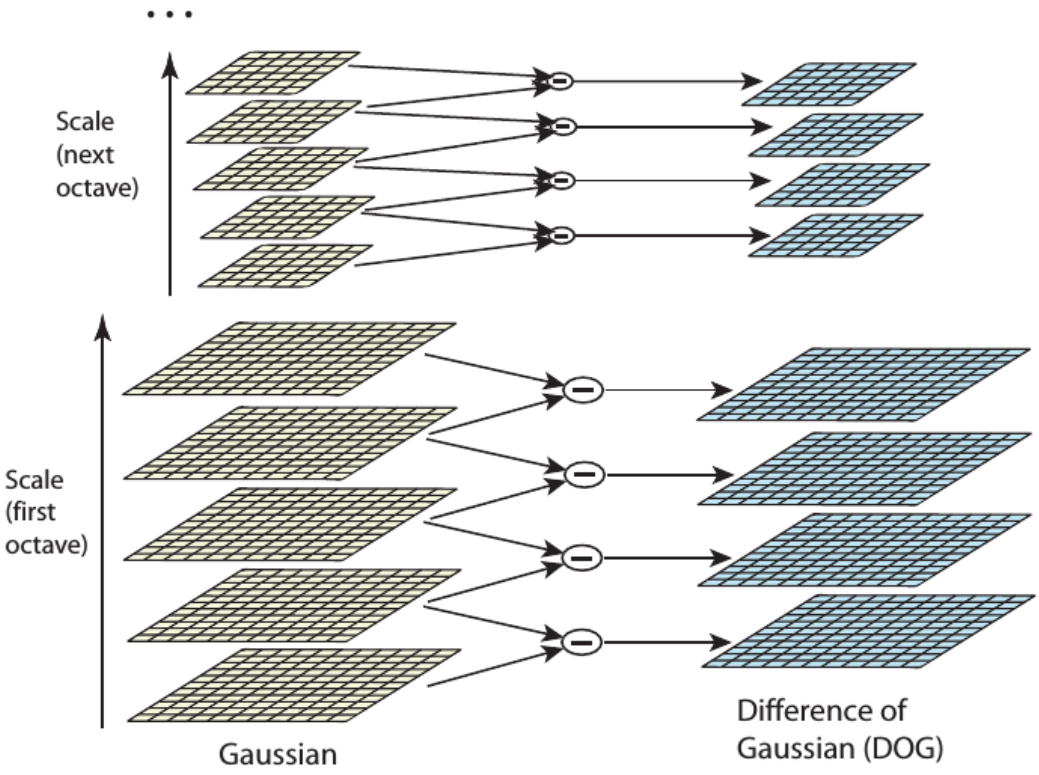


FIGURE 1.1 – Différence de Gausienne [11]

Un point d'intérêt candidat (x, y, σ) est défini comme un point où un extremum du DoG est atteint par rapport à ses voisins immédiats, c'est-à-dire sur l'ensemble contenant 26 autres points défini par :

$$\{D\left(x+\delta_x, y+\delta_y, s\sigma\right), \delta_x \in \{-1,0,1\}, \delta_y \in \{-1,0,1\}, s \in \{k^{-1},1,k\}\}$$

On peut voir l'image ci-dessous pour être facile à comprendre

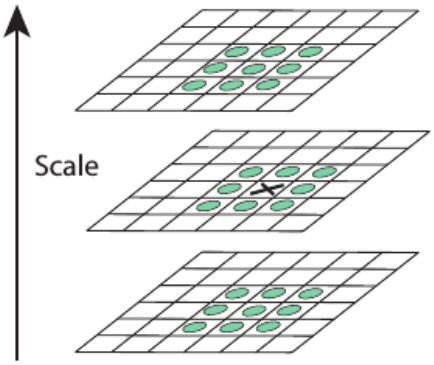


FIGURE 1.2 – [11] Le maxima et le minima des images de différence de gaussienne sont détectés en comparant un pixel (marqué X) à ses 26 voisins dans les régions de 3x3 aux échelles actuelles et adjacents (marqué avec des cercles).

1.3.3 Localisation précise de points d'intérêt

L'étape de détection d'extremums produit en général un grand nombre de points-clés candidats, dont certains sont instables. De plus, leur localisation, en particulier aux échelles les plus grandes (autrement dit dans les octaves supérieures de la pyramide où la résolution est plus faible) reste approximative. De ce fait, des traitements supplémentaires sont appliqués, pour un objectif double : d'une part, reconverger la position des points pour améliorer la précision sur x, y et σ, d'autre part, éliminer les points de faible contraste ou situés sur des arêtes de contour à faible courbure et donc susceptibles de "glisser" facilement.

1.3.4 Assignation d'orientation

L'étape d'assignation d'orientation consiste à attribuer à chaque point-clé une ou plusieurs orientations déterminées localement sur l'image à partir de la direction des gradients dans un voisinage autour du point. Dans la mesure où les descripteurs sont calculés relativement à ces orientations, cette étape est essentielle pour garantir l'invariance de ceux-ci à la rotation : les mêmes descripteurs doivent pouvoir être obtenus à partir d'une même image, quelle qu'en soit l'orientation[11].

Pour un point-clé donné (x_0, y_0, σ_0), le calcul s'effectue sur $L(x, y, \sigma_0)$, à savoir le gradient de la pyramide dont le paramètre est le plus proche du facteur d'échelle du point. De cette façon, le calcul est également invariant à l'échelle. À chaque position dans un voisinage du point-clé, on estime le

gradient par différences finies symétriques, puis son amplitude (c.-à-d. sa norme) $m(x, y)$, et son orientation $\theta(x, y)$ [11] :

$$m\left(x,y\right)=\sqrt{\left(L\left(x+1,y\right)-L\left(x-1,y\right)\right)^{2}+\left(L\left(x,y+1\right)-L\left(x,y-1\right)\right)^{2}} \tag{1.3}$$

$$\theta\left(x,y\right)=\tan^{-1}\left(\frac{L\left(x,y+1\right)-L\left(x,y-1\right)}{L\left(x+1,y\right)-L\left(x-1,y\right)}\right)\forall\left(x,y\right)\ \text{dans un voisinage de}\ \left(x_0,y_0\right) \tag{1.4}$$

Un histogramme des orientations sur le voisinage est réalisé avec 36 intervalles, couvrant chacun 10 degrés d'angle. L'histogramme est doublement pondéré : d'une part, par une fenêtre circulaire gaussienne de paramètre égal à 1,5 fois le facteur d'échelle du point-clé σ_0, d'autre part, par l'amplitude de chaque point. Les pics dans cet histogramme correspondent aux orientations dominantes. Toutes les orientations dominantes permettant d'atteindre au moins 80% de la valeur maximale sont prises en considération, ce qui provoque si nécessaire la création de points-clés supplémentaires ne différant que par leur orientation principale[11].

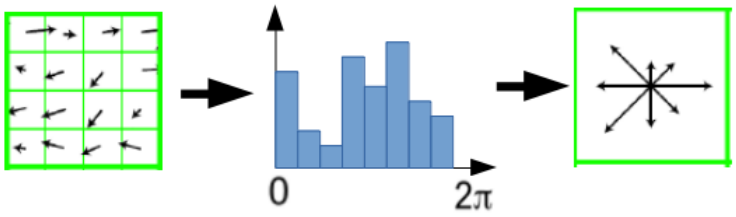


FIGURE 1.3 – Illustration de la construction de l'histogramme des orientations

À l'issue de cette étape, un point-clé est donc défini par quatre paramètres (x, y, σ, θ). Il est à noter qu'il est parfaitement possible qu'il y ait sur une même image plusieurs points-clés qui ne différent que par un seul de ces quatre paramètres (le facteur d'échelle ou l'orientation, par exemple).

1.3.5 Descripteur de point d'intérêt

Une fois les points-clés, associés à des facteurs d'échelles et à des orientations, détectés et leur invariance aux changements d'échelles et aux rotations assurée, arrive l'étape de calcul des vecteurs descripteurs, traduisant

numériquement chacun de ces points-clés. À cette occasion, des traitements supplémentaires vont permettre d'assurer un surcroît de pouvoir discriminant en rendant les descripteurs invariants à d'autres transformations telles que la luminosité, le changement de point de vue 3D, etc. Cette étape est réalisée sur l'image lissée avec le paramètre de facteur d'échelle le plus proche de celui du point-clé considéré[11].

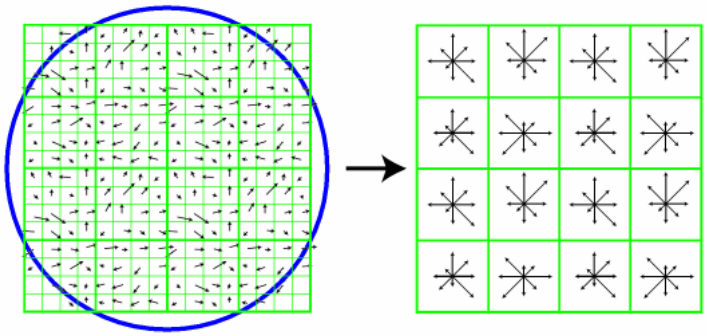


FIGURE 1.4 – Construction d'un descripteur SIFT[4]

Autour de ce point, on commence par modifier le système de coordonnées local pour garantir l'invariance à la rotation, en utilisant une rotation d'angle égal à l'orientation du point-clé, mais de sens opposé. On considère ensuite, toujours autour du point-clé, une région de $16x16$ pixels, subdivisée en $4x4$ zones de $4x4$ pixels chacune. Sur chaque zone est calculé un histogramme des orientations comportant 8 intervalles. En chaque point de la zone, l'orientation et l'amplitude du gradient sont calculés comme précédemment. L'orientation détermine l'intervalle à incrémenter dans l'histogramme, ce qui se fait avec une double pondération par l'amplitude et par une fenêtre gaussienne centrée sur le point clé, de paramètre égal à 1,5 fois le facteur d'échelle du point-clé[11].

Ensuite, les 16 histogrammes à 8 intervalles chacun sont concaténés et normalisés. Dans le but de diminuer la sensibilité du descripteur aux changements de luminosité, les valeurs sont plafonnées à 0,2 et l'histogramme est de nouveau normalisé, pour finalement fournir le descripteur SIFT du point-clé, de dimension 128.

1.4 Modèle BoVW (Bag of visual word)

La représentation par le sac de mots visuels *(ou bag of visual words en anglais)* est une description de document (texte, image, ...) très utilisée

en recherche d'information. Spécialement, dans la classification d'images, ce modèle est largement utilisé après l'étape d'extraction des descripteurs. Dans BovW, les méthodes de cluster sont utilisées pour grouper des descripteurs. Actuellement, la méthode K-moyenne (K-means)[16] est beaucoup utilisée pour grouper des des descripteurs SIFT aux clusters, chaque cluster est considéré comme un mot visuel, l'ensemble de mots visuels jouent le rôle d'un dictionnaire de mots visuels. Ensuite, BovW met chaque descripteur de chaque image au cluster qui est le plus proche(se base sur la distance entre chaque descripteur et sa future cluster). Suite, chaque image est décrite comme un histogramme des mots. Ce modèle est décrite comme l'image ci-dessous.

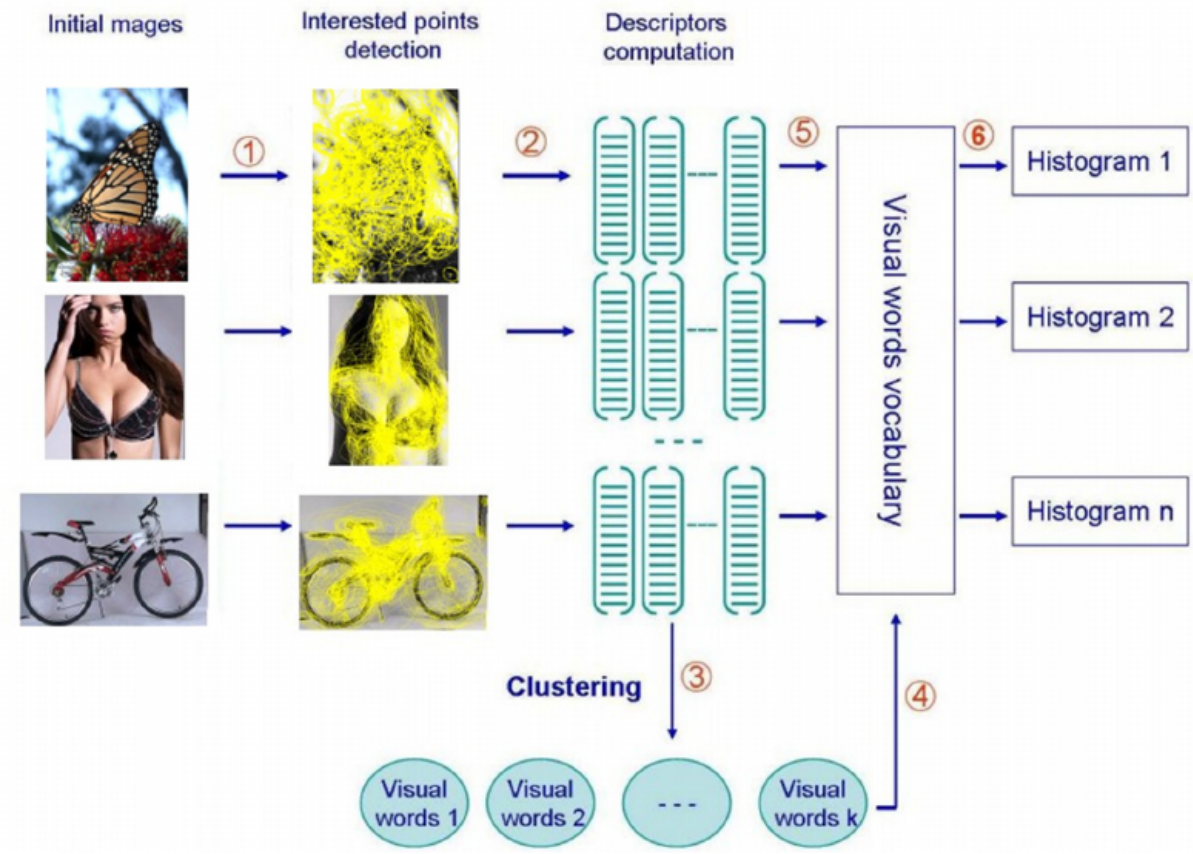


FIGURE 1.5 – Model de BOW [9]

Chapitre 2

Apprentissage automatique

2.1 Introduction

L'apprentissage automatique (machine learning) est un domaine de l'intelligence artificielle, qui permet aux machines d'apprendre à partir des données. L'apprentissage automatique est la discipline scientifique concernée par le développement, l'analyse et l'implémentation de méthodes automatisables qui permettent à une machine d'évoluer autonome-ment grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques. Le problème ici est que la donnée d'observation est petite, elle ne peut pas comprendre tout l'ensemble de données d'entrées (tous les cas) qui est trop grande. Un programme d'apprentissage automatique doit généraliser des données limite afin de donner des réactions intelligentes sur des nouveaux exemples.

Dans la classification d'images, les images d'apprentissage ne peut pas se compose tous les cas de chaque image, donc, les algorithmes classiques ne permettent pas de reconnaitre des images. C'est la raison pour laquelle on a besoin des méthodes intelligentes comme l'apprentissage automatique pour prédire des nouvelles images entrées. Après avoir appliqué SIFT et BoVW, les images deviennent des chiffres. L'apprentissage automatique utilise ces chiffres pour la classification.

2.2 Méthode SVM (Support Vector Machine)

Les machines à vecteurs de support (Support Vector Machine, SVM) sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de classification et de régression. Dans cette section, nous ne par-

lons que SVM pour la classification concernant notre projet dans la pratique.

Supposons que l'on a le problème de classification. Le cas simple est le cas d'une fonction de classification linéaire comme dans l'image 2.1. On a m exemples d'entrées $x_1, x_2, ..., x_m$ dans l'espace de N dimensions. Chaque exemple x_i a un label $y_i : y_1, y_2, ..., y_m$ ($y_i \in \{-1, 1\}$). Plusieurs hyperplans peuvent séparer ces 2 classes, quel est l'hyperplan optimal ?

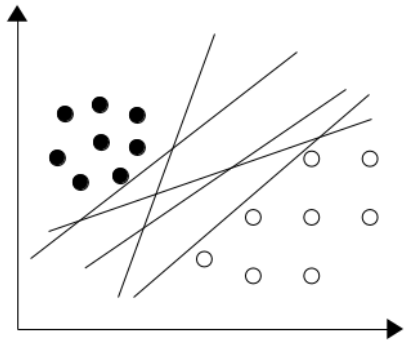


FIGURE 2.1 – Classification linéaire

SVM cherche l'hyperplan optimal qui est défini comme l'hyperplan qui maximise la marge entre les échantillons et l'hyperplan séparateur. La marge est la distance entre l'hyperplan et les échantillons les plus proches. Ces derniers sont appelés *vecteurs de supports*. Dans l'espace de N dimensions, l'hyperplan est défini par le vecteur $w = [w_1, w_2, ..., w_n]$ et b. SVM cherche l'hyperplan (w, b) pour classifier les données comme l'image 2.2. Il existe des raisons théoriques à ce choix. Vapnik a montré que la capacité des classes d'hyperplans séparateurs diminue lorsque leur marge augmente[21].

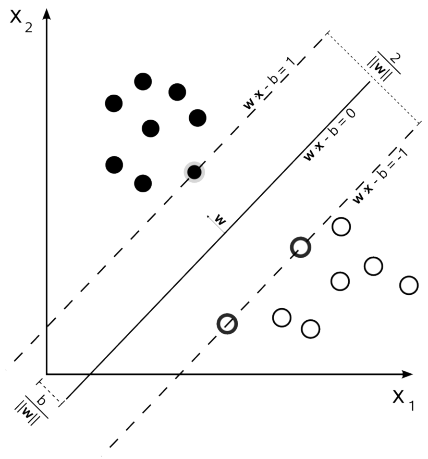


FIGURE 2.2 – L'hyperplan optimal

Pour trouver l'hyperplan optimal, tout d'abord on construit 2 supports hyperplans $x^T.w - b = 1$ et $x^T.w - b = -1$. L'hyperplan optimal est trouvé au milieux de ces deux hyperplans. Après avoir trouvé w et b, l'hyperplan optimal est défini : $x^T.w - b = 0$ et Par rapport à 2 supports hyperplans parallèles (voir l'image 2.2), la classification est réalisée grâce aux 2.1 et 2.2.

$$x_i.w - b \geq +1, \Rightarrow y_i = 1 \tag{2.1}$$

$$x_i.w - b \leq -1, \Rightarrow y_i = -1 \tag{2.2}$$

La classification peut être expliquée : l'hyperplan supporte la classe ***(+1)*** est l'hyperplan que les points qui a le label ***(+1)*** au dessus de l'hyperplan. Comme ça, l'hyperplan supporte la classe ***(-1)*** est l'hyperplan que les points qui a le label ***(-1)*** au dessous de l'hyperplan. Par rapport aux 2.1 et 2.2, on a le formule 2.3

$$y_i.(x_i.w - b) \geq 1 \tag{2.3}$$

Dans le cas où l'algorithme ne peut pas trouver (w, b) satisfait le problème (les données sont inséparable), nous devons accepter des erreurs z_i. Chaque exemple i est dans son vrais hyperplan, donc, son erreur $z_i = 0$, sinon, son erreur z_i est défini par la distance entre cet exemple et son hyperplan. Le formule 2.3 devient 2.4 :

$$y_i.(x_i.w - b) + z_i \geq 1 \tag{2.4}$$

Nous trouvons que la classification est facile si l'on a trouvé w et b. La difficulté de la méthode SVM est que comment trouver w et b. Cette tâche est réalisée après avoir trouvé la solution du programme de quadratique 2.5.

$$\begin{aligned} \min \quad \Psi(w,b,z) &= \frac{1}{2}||w||^2 + c.\sum_{i=1}^{m} z_i \\ \text{s.t} \quad y_i.(x_i.w - b) + z_i &\geq 1 \\ \text{and} \quad z_i &\geq 0 \end{aligned} \tag{2.5}$$

Le problème d'optimisation quadratique 2.5 est un des problèmes d'optimisation qui est normalement recherché dans le domaine de mathématique d'optimisation. Pour l'implémentation ce problème, [17] et [18] ont la complexité de $O(N^2)$[9] (N est le nombre d'exemples). Ce sont des programmes les plus utilisés actuel.

2.3 Méthode SVM avec SGD (Stochastic gradient descent)

2.3.1 Descente de gradient

Le SVM standard est efficace mais la complicité est grande ($O(N^2)$), donc, c'est intraitable pour les larges données. La méthode SVM avec la descente de gradient est créées pour résoudre ce problème. Au lieu de chercher la solution pour résoudre le problème du programme de quadratique 2.5 comme la méthode SVM, cette méthode utilise la descente de gradient pour trouver w et b qui minimise Ψ.

Pour un programme quadratique $f(x)$, la descente de gradient peut être décrite comme l'image ci-dessous :

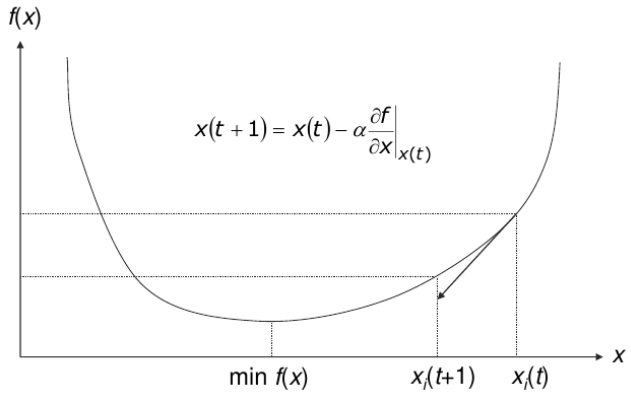


FIGURE 2.3 – Descente de gradient

L'algorithme du gradient désigne un algorithme d'optimisation différentiable. Il est par conséquent destiné à minimiser une fonction réelle différentiable définie sur un espace euclidien (par exemple, R^n, l'espace des n-dimensions de nombres réels, muni d'un produit scalaire) ou, plus généralement, sur un espace hilbertien (de dimension infinie). L'algorithme est itératif et procède donc par améliorations successives. Au point courant, un déplacement est effectué dans la direction opposée au gradient, de manière à faire décroître la fonction. Le déplacement le long de cette direction est déterminé par la technique numérique connue sous le nom de recherche linéaire. Pour la méthode descente de gradient, on utilise tous les pairs (x, y) pour calculer le sous-gradient. La méthode descente de gradient stochastique utilise moins exemples chaque itération pour calculer le sous-gradient.

2.3.2 Descente de gradient stochastique (SGD)

Dans la méthode SGD, au lieu d'utiliser tous les pairs (x, y) pour calculer le sous-gradient, on peut utiliser un ou quelques exemples aléatoires. Dans Pegasos, les auteurs ignore b dans le formule 2.5. Le contraint $y_i.(x_i.w - b) + z_i \geq 1$ peut être remplacé par :

$$z_i \geq 1 - y_i.(x_i.w) \tag{2.6}$$

A partir du contraint 2.6 et $z_i \geq 0$, on peut écrire la function de perte :

$$z_i = max\{0, 1 - y_i.(x_i.w)\} \tag{2.7}$$

Donc, le programme de quadratique 2.5 peut remplacer par le formule 2.8 :

$$\min \quad \Psi(w, x, y) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} max\{0, 1 - y_i.(x_i.w)\} \tag{2.8}$$

L'implémentation Pegasos de la méthode SGD est proposée dans [2] prend un seul exemple pour la mise à jour de w. Donc, dans chaque étape t, on a $\Psi(w, i_t)$ comme le formule 2.9

$$\min \quad \Psi(w, i_t) = \frac{1}{2}||w||^2 + max\{0, 1 - y_i.(x_i.w)\} \tag{2.9}$$

Dans cette méthode, w est mise à jour en T étapes avec une vitesse d'apprentissage η_t. A chaque étape t, SGD prend un exemple (x_i, y_i) aléatoire pour calculer le sous-gradient et met à jour w_{t+1}.

$$w_{t+1} = w_t - \eta_t.\nabla_w \Psi(w, i_t) \tag{2.10}$$

Où

$$\nabla_w \Psi(w, i_t) = \lambda.w_t - \Bbbk[y_i.(w_t, x_{it}) < 1].y_{it}.x_{it}$$

et

$$\eta_t = \frac{1}{\lambda.t}$$

$\Bbbk[y_i.(w_t, x_{it}) < 1]$ prend le value 1 si $y_i.(w_t, x_{it}) < 1$ et il prend le value 0 si inverse

Donc, le formule 2.10 devient

$$w_{t+1} = (1 - \frac{1}{t})w_t + \frac{1}{\lambda.t}\Bbbk[y_i.(w_t, x_{it}) < 1].y_{it}.x_{it} \tag{2.11}$$

2.3.3 Mini-batch interaction

Pour plus générale, dans Pegasos, les auteurs ne choisissent pas un seul exemple, mais ils choisissent k exemples dans chaque étape t. Pour cette modification, w est mise à jour comme dans le formule 2.12

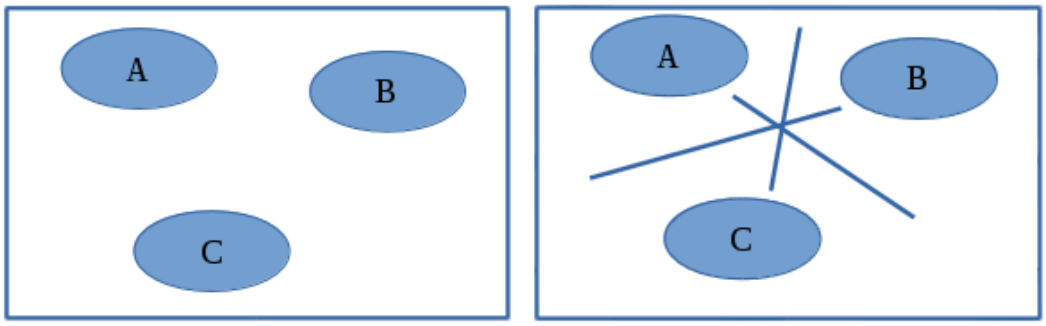
$$w_{t+1} = (1 - \frac{1}{t})w_t + \frac{1}{\lambda.t.k}\sum_{i=1}^{k}\Bbbk[y_i.(w_t, x_{it}) < 1].y_{it}.x_{it} \tag{2.12}$$

2.4 Méthode MC-SGD (Multi Class - Stochastic gradient descent)

Plupart d'algorithme de SVM ne traite que le problème de 2 classes (binaire). Il existe plusieurs extensions d'une classification binaire afin de traiter le problème de multi-class classification. Actuellement, on a deux façons pour traiter ce problème. L'une considère directement à résoudre le problème de multi-classes [19]. L'autre divise d'abord le problème en plusieurs problèmes de 2 classes et chaque problème de 2 classes est traité par la classification binaire comme one-versus-one [20] et one-versus-all [22]. En pratique, one-vs-one et one-vs-all sont beaucoup utilisés par leur simple.

2.4.1 One-versus-one

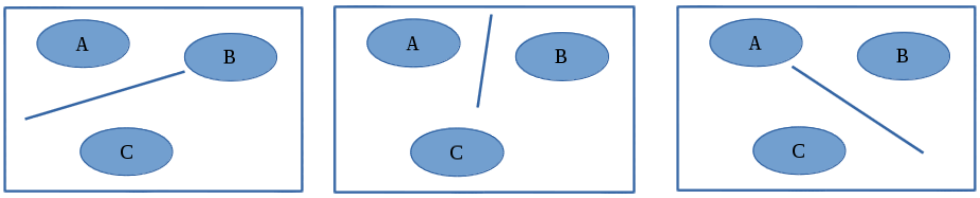
One-vs-one construit *k(k-1)/2* (k est le nombre de classes) classificateurs, c'est à dire, il utilise tous les paires de classes différents pour l'apprentissage. La prédiction est réalisée par voter et la majorité classificateur va être choisi et la classe correspondant est la classe de prédiction.



(a) Problème (b) one-vs-one

FIGURE 2.4 – Problème de multi-classes one-versus-one

L'image 2.4b est comme un résumé de one-vs-one. Pour plus claire, nous vous listons les trois classificateurs de ces trois classes comme l'image 2.5

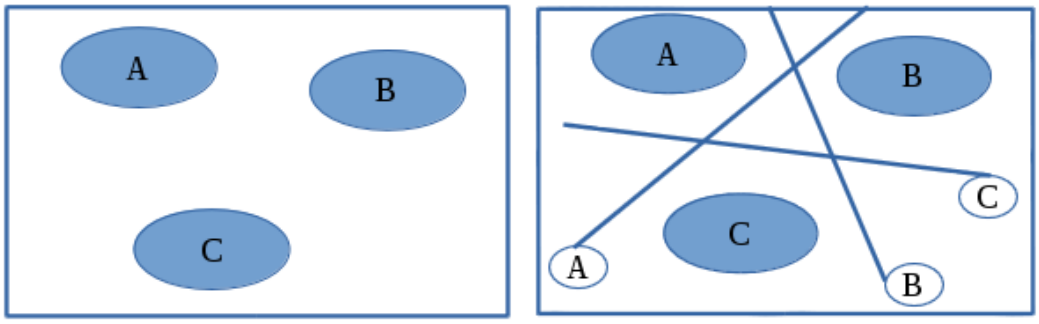


(a) classe A vs classe C (b) classe A vs classe B (c) classe B vs classe C

FIGURE 2.5 – Problème de multi-classes one-versus-one détaillé

2.4.2 One-versus-all

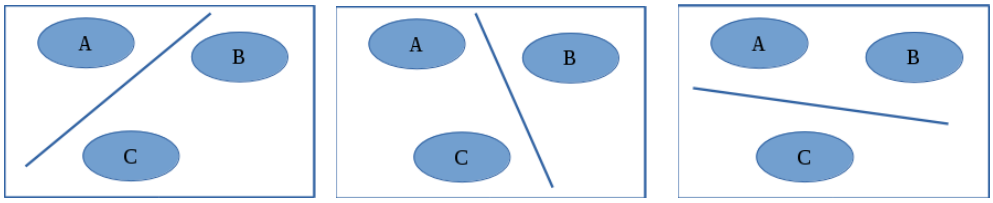
One-vs-all construit k classificateurs, à chaque classificateur c, il divise la classes c contre tous les restes. La classe de prédiction est la classe ayant la distance la plus courte entre son classificateur et l'exemple d'entrée.



(a) Problème (b) one-vs-all

FIGURE 2.6 – Problème de multi-classes one-versus-all

L'image 2.6b est comme un résumé de one-vs-all. Pour plus claire, nous vous listons les trois classificateurs de ces trois classes comme l'image 2.7



(a) classe A (b) class B (c) class C

FIGURE 2.7 – Problème de multi-classes one-versus-all détaillé

Ces deux méthodes souffrent toutes de deux défauts. La version One-vs-one construit beaucoup de classificateurs. Dans la version one-versus-all, rien n'indique que les valeurs du résultat de classification des k classificateurs soient comparables. De plus le problème n'est plus équilibré, par exemple avec k = 10, on utilise seulement 10% d'exemples positifs pour 90% d'exemples négatifs.

Chapitre 3

Algorithme MC-SGD pour la classification d'images

3.1 Introduction

Dans le chapitre précédent, nous avons généralement présenté des méthodes concernant notre travail. Ce chapitre permet de présenter le détail de ces méthode et essayé d'écrire des pseudo-codes de ces algorithme.

Comme nous avons parlé, notre processus comprend 2 stages : extraction des caractéristiques et apprentissage automatique. Tout d'abord nous présentons le stage d'extraction des caractéristiques avec le modèle sac de mots. Ensuite nous parlons du stage d'apprentissage automatique avec l'algorithme MC-SGD que nous avons proposé.

3.2 Représentation d'une image par des descripteurs et le modèle sac de mots

La description des images est très importante dans la classification d'images. Cette étape beaucoup influence au résultat final. Dans le domaine de recherche d'images et de classification d'images, le descripteur SIFT [11] est un caractéristique important pour la représentation des images. Cette méthode est de plus en plus populaire. A partir de l'idée de la classification de textes, dans la recherche en 2007 [23], les auteurs proposent un système de classification d'images utilisant le descripteur SIFT et le modèle sac de mots visuels (Bag of Visual Word). Cette méthode peut diviser en 3 étapes :

1. Trouver des descripteurs locales des images (des caractères)

2. Construire un dictionnaire a partir des caractères

3. Représenter des images par des histogrammes

Dans ces étapes, nous avons réutilisé des codages existant en modifiant des entrées et des sorties pour adapter aux étapes suivantes. Précisément, dans l'étape *1*, nous avons réutilisé le codage de D.Lowe [1] pour créer des vecteurs de caractéristiques (descripteurs). Ensuite, l'étape *2* sert à construire un dictionnaire. Dans cette étape, la méthode k-moyenne [24], l'implémentation de Nguyen-Khang PHAM [25] est appliquée afin de créer des clusters à partir des vecteurs SIFT créés dans l'étape *1*. L'ensemble de clusters est considéré comme un dictionnaire. Enfin, dans l'étape *3*, une image est représentée par un histogramme. Dans cette étape, d'abord, on met chaque vecteur SIFT dans chaque image à un cluster le plus proche de ce vecteur (basé sur la distance entre ce vecteur au centre des clusters). Ensuite, on compte le fréquence des mots d'une image existe dans le dictionnaire créé dans l'étape *2* pour représenter l'image comme un histogramme de fréquence. Pour l'étape *3*, nous avons implémenté en C/C++.

3.3 Apprentissage automatique

Avec notre processus, après avoir apliqué le modèle sac de mot visuel, nous appliquons un algorithme d'apprentissage pour classifier des images vers leurs classes. Nous avons développé l'algorithme MC-SGD et pour vérifier la performance de notre algorithme, nous avons fait la comparaison avec la bibliothèque LibSVM [18], l'implémentation de la méthode SVM la plus utilisée actuellement.

3.3.1 Descente de gradient stochastique (SGD)

Pour cette méthode, il existe plusieurs d'implémentations différentes. Dans notre travail, nous avons réutilisé l'implémentation *Pegasos* [2] ci-dessous :

Algorithm 1 L'algorithm d'apprentissage SGD binaire

procedure TRAINBINAIRESGD(D, y, λ, T, n)
    Initialiser $w_1 = 0$
    for `t = 0; t < T; t++` **do**
        $\eta = \frac{1}{\lambda}$
                ▷ Boucle pour choisir ex exemples chaque cycle
        for `i = 0; i < n; i++` **do**
            `choisir un exemple aléatoire dans D`
            if $y_{i_t}.(w_t, x_{it}) < 1$ **then**
                $w_{t+1} = (1 - \eta_t.\lambda).w_t + \eta_t.y_{i_t}.x_{i_t}$
            else
                $w_{t+1} = (1 - \eta_t.\lambda).w_t$
    Return w_t

3.3.2 Descente de gradient stochastique pour multi-classe (MC-SGD)

Comme les autres algorithme de la méthode SVM, le SGD est implémenté pour résoudre le problème de classification de 2 classes. L'implémentation *Pegasos* [2] est efficace. A partir de cette implémentation, nous l'avons modifiée et développée pour résoudre le problème de multi-classes (k classes, $k \geq 3$).

Pour le problème de classification multi-classes, nous avons implémenté les deux versions *one-vs-one et one-vs-all* du MC-SGD.

One-vs-one

Pour la classification de k classes, cette méthode construit k(k-1)/2 classificateurs. Par exemple, quand le problème a 1000 classes, *one-vs-one* va construire 450000 classificateurs. On voit ci-dessous l'implémentation de MC-SGD d'après one-vs-one.

Algorithm 2 L'algorithm d'apprentissage MC-SGD one-vs-one

1: **procedure** TRAINMCSGDONEONE(D, y, k, λ, T, n)
▷ D est les données d'apprentissage
2: ▷ y est les labels des données
3: ▷ k est le nombre de classes dans D
4: ▷ λ est le constant possible
5: ▷ T est le nombre de cycles
6: ▷ n est le nombre d'exemple par cycle
7:
8: Initialiser k(k-1)/2 modèles w
9: **for** `c1 = 0; c1 < k-1; c1++` **do**
10: **for** `c2 = c1+1; c2 < k; c2++` **do**
11: ▷ Préparation des données pour 1-vs-1
12: $y_i = +1$ `si l'exemple de classe c1,` $y_i = -1$ `si de c2`
13: $w_c = trainBinaireSGDSVM(D, y, \lambda, T, n)$ $(c1 - vs - c2)$
14: Return w

One-vs-all

One-vs-one est balancée, la classification est acceptable mais cet algorithme construit beaucoup de classificateurs k(k-1)/2. quand le nombre de classes est grand, cet algorithme perd beaucoup de temps pour l'apprentissage et la classification.

Pour la classification de k classes, one-vs-all ne construit que k classificateurs. Par exemple, quand le problème a 1000 classes, *one-vs-all* va construire 1000 classificateurs. On trouve que 1000 est beaucoup plus petit que 450000. On voit ci-dessous l'implémentation du MC-SGD de la version one-vs-all.

Algorithm 3 L'algorithm d'apprentissage MC-SGD one-vs-all

1: **procedure** TRAINMCSGDONEALL(D, y, k, λ, T, n)
▷ D est les données d'apprentissage
2: ▷ y est les labels des données
3: ▷ k est le nombre de classes dans D
4: ▷ λ est le constant possible
5: ▷ T est le nombre de cycles
6: ▷ n est le nombre d'exemple par cycle
7:
8: Initialiser k modèles w
9: **for** `c = 0; c < k; c++` **do**
10: ▷ Préparation des données pour one-vs-all
11: $y_i = +1$ `si l'exemple de classe c,` $y_i = -1$ `sinon`
12: $w_c = trainBinaireSGD(D, y, \lambda, T, n)$ $(c - vs - all)$
13: Return w

Problème de données non balancées du MC-SGD

Nous trouvons que notre processus est acceptable, one-versus-all s'adapte bien pour la classification multi-classes. Malheureusement, la version one-vs-all essaie de séparer un ensemble de k exemples vers une classe positive et (k-1) classes négatives, donc, pour les bases de données qui se composent beaucoup de classes, cet algorithme trouve le problème de données non balancées. Par exemple, si le problème de 1000 classes, on a 1 exemple positif et 999 exemple négatifs. Chaque fois où le SGD choisit aléatoire un exemple pour apprendre, les exemples positifs sont rarement choisis (probabilités de *0.1%*) mais les exemples négatifs sont très souvent choisis (probabilités de *99.9%*). Donc, le résultat de classification est influencé ! Pour ce problème, on a deux options pour le résoudre : optimisation par équilibration des données et optimisation par algorithme.

Optimisation par équilibration des données

La version d'optimisation par équilibration des données de l'option one-vs-all consiste à augmenter la probabilité sélectionné des exemples positifs pour que la probabilité sélectionnée des exemples positifs soit égale celle des exemples négatifs. Pour cela, au lieu de choisir aléatoirement un exemple pour d'apprendre, l'algorithme choisit aléatoirement entre -1 et +1, si +1 est choisi, le SGD choisit aléatoirement un exemple dans l'ensemble d'exemple positif et inverse.

Algorithm 4 L'algorithm d'apprentissage SGD binaire balancé

procedure TRAINBINAIRESGDSVM(D, y, λ, T, n)
    Initialiser $w_1 = 0$
    for `t = 0; t < T; t++` **do**
        $\eta = \frac{1}{\lambda}$
                ▷ Boucle pour choisir n exemples chaque cycle
        for `i = 0; i < n; i++` **do**
            `choisir aléatoirement entre +1 et -1`
            if $+1$ est choisi **then**
                `choisir un exemple positive aléatoire dans D`
            else
                `choisir un exemple négative aléatoire dans D`
            if $y_{it}.(w_t, x_{it}) < 1$ **then**
                $w_{t+1} = (1 - \eta_t.\lambda).w_t + \eta_t.y_{it}.x_{it}$
            else
                $w_{t+1} = (1 - \eta_t.\lambda).w_t$
    Return w_t

Équilibration par optimisation de l'algorithme

Au lieu d'augmenter la probabilité sélectionné des exemples positifs comme l'équilibration par optimisation des données, la version d'optimisation par optimisation de l'algorithme de l'option one-vs-all consiste à mettre à jours le poids avec le coût différent dans deux cas différents : petit erreur et grande erreur. Particulièrement, on met deux seuils de 0 et de 1 pour $y_{it}.(w_t.x_{it}))$.

Algorithm 5 L'algorithm d'apprentissage SGD binaire balancé

procedure TRAINBINAIRESGDSVM(D, y, λ, T, n)
    Initialiser $w_1 = 0$
    for `t = 0; t < T; t++` **do**
        $\eta = \frac{1}{\lambda}$
                ▷ Boucle pour choisir n exemples chaque cycle
        for `i = 0; i < n; i++` **do**
            `choisir un exemple aléatoire dans D`
            if $y_{it}.(w_t, x_{it}) < 0$ **then**
                $w_{t+1} = (1 - \eta_t.\lambda).w_t + \frac{\eta_t.y_{it}.x_{it}}{D+}$
            else if $y_{it}.(w_t, x_{it}) < 1$ **then**
                $w_{t+1} = (1 - \eta_t.\lambda).w_t + \frac{\eta_t.y_{it}.x_{it}.(1-y_{it}.(w_t.x_{it}))}{D+}$
            else
                $w_{t+1} = (1 - \eta_t.\lambda).w_t$
    Return w_t

Parallélisation du MC-SGD

Nous trouvons que l'algorithme 3 linéaire apprend les modèles l'un après l'autre sur un seul cœur du processeur d'un ordinateur. Donc, les autres cœurs ne font rien et l'algorithme est lent. A notre époque, nous pouvons utiliser tous les cœurs possibles pour améliorer la vitesse de notre algorithme en utilisant la programmation parallèle.

Pour le problème de multi-classes (k classes), l'algorithme SGD apprend indépendant chaque classificateur. Donc, c'est possible d'apprendre chaque classificateur sur chaque cœur différent. Dans notre implémentation, nous avons parallélisé l'itération sur l'apprentissage des classificateur. Supposons que nous avons p processeurs et k classificateurs. Donc, chaque processeur apprend $n = \frac{k}{p}$ ou $n = \frac{k}{p} + 1$ classificateurs.

Algorithm 6 L'algorithm d'apprentissage SGD parallèle pour multi-classes

1: **procedure** TRAINMCSGDPARALLEL(D, y, k, λ, T, n)
2: Initialiser k modèles w
3:
4: **Pocesseur 1 :**
5: $y_i = +1$ `si l'exemple de classe c (c = 1, classe 1.p + 1, classe 2.p + 1, ...) et` $y_i = -1$ `sinon`
6: $w_c = trainBinaireSGD(D, y, \lambda, T, n)$ $(c - vs - all)$
7:
8: **Pocesseur 2 :**
9: $y_i = +1$ `si l'exemple de classe c (c = 2, classe 1.p + 2, classe 2.p + 2, ...) et` $y_i = -1$ `sinon`
10: $w_c = trainBinaireSGD(D, y, \lambda, T, n)$ $(c - vs - all)$
11: .
12: .
13: .
14:
15: **Pocesseur p :**
16: $y_i = +1$ `si l'exemple de classe c (c = p, classe 1.p + p, classe 2.p + p, ...) et` $y_i = -1$ `sinon`
17: $w_c = trainBinaireSGD(D, y, \lambda, T, n)$ $(c - vs - all)$
18: Return w

Avec la structure comme l'algorithme 6, à chaque boucle, p processeurs apprennent p classificateurs en parallèle. Alors, la vitesse améliore presque p fois si l'on compare avec la version linéaire de cet algorithme, l'algorithme 3.

3.3.3 MC-SGD-Toy pour MC-SGD

Se basant sur SVM-Toy[18], nous développons une interface pour la démonstration de MC-SGD.
Données :

- Chaque point est un exemple
- x, y sont 2 caractéristiques d'exemples

On applique MC-SGD pour la classification des points entrées par rapport des distances entre eux.

Comme nous avons présenté dans la partie de théorie, le SGD ignore b dans le formule 2.5. C'est à dire, les classificateurs de notre algorithme ne

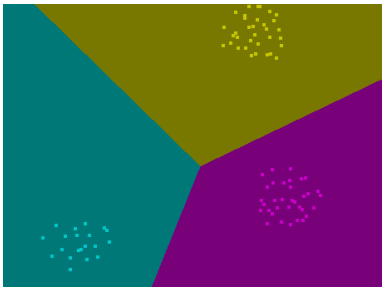
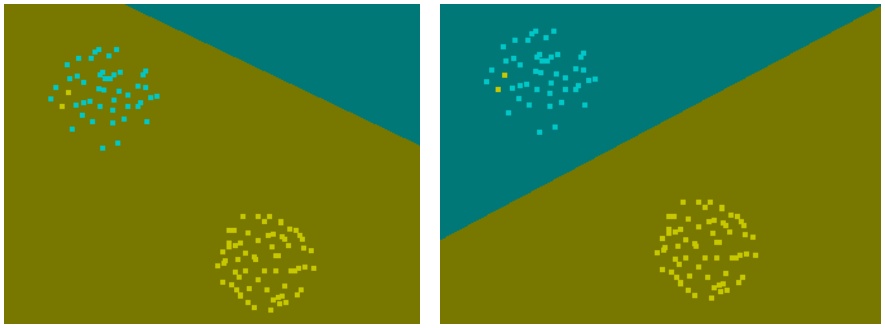


FIGURE 3.1 – MC-SGD-Toy

passe que l'origine des coordonnées cartésiennes. Donc, la classification dans l'espace de deux dimension comme les coordonnées cartésiennes n'est pas bien quand l'entrée comme le figure 3.2a.



(a) MC-SGD-Toy origine (b) MC-SGD-Toy non origine

FIGURE 3.2 – MC-SGD-Toy pour problème non origine

Le figure 3.2b est le résultat de classification de MC-SGD après la modification. L'idée de cette modification est d'ajouter une troisième dimension dans chaque exemple. Un point p(x,y) devient p(x,y,1). Nous trouvons clairement sur le figure 3.2 que la modification est mieux que le MC-SGD origine car c'est plus générale.

Chapitre 4

Expérimentation

4.1 Introduction

Dans ce chapitre nous parlons des logiciels, matériels et des données que nous utilisons pour implémenter et tester notre algorithme. Ensuite, nous présentons le résultat de notre algorithme sur ces données. Nous conclurons aussi notre travail à la fin de ce chapitre.

4.2 Logiciels et matériels

Dans ce stage, nous avons utilisé des logiciels et des matériels listées ci-dessous :
Logiciels

- Extraction des caractéristiques : SIFT[low99], K-Means(pour clustering)
- Méthode pour la classification : SGD binaire (implémentation Pegasos), LibSVM(pour la comparaison)
- En C/C++, sous GNU-Linux

Matériels

- PC core-i7, 8 cœurs, 3G RAM
- Système d'exploitation : GNU-Linux Fedora 20

Nos propositions

- Classification multi-classes : MC-SGD et MC-SGD parallèle, implémentés en C/C++ en utilisant l'OpenMP

- Outils pour étudier notre algorithme : MC-SGD-Toy, basé sur SVM-Toy[18]

4.3 Jeux de données

Les données sont traitées à partir des bases d'images en utilisant la méthode SIFT avec le nombre de dimension et le modèle BoVW avec le nombre de mots visuels dans la colonne 4 et 5 dans la table ci-dessous.

Données	#classes	#exemples	#dimension(SIFT)	#mots(BoVW)
Cal 101	101	1515	128	124000
Cal 7 3D	7	4290	128	5000
ImgNet 3d	10	4450	128	5000
ImgNet	10	4450	128	50000

TABLE 4.1 – Information sur des données

4.4 Méthode SGD binaire

Pour cette étape, nous voulons comparer le SGD binaire avec la LibSVM pour voir si quelle méthode est plus efficace. Nous utilisons ci-dessous la base de données binaire (2 classes) de la LibSVM [26], Adult avec la taille d'exemple d'entrées différente.

Dans cette expérimentation, les données sont linéaires. C'est la raison pour laquelle nous avons utilisé le SVM linéaire car elle est plus efficace que les autres fonctions de noyau. Pour l'algorithme SGD binaire, nous avons utilisé $T = 10000$ itérations et dans chaque itération nous choisissons 10 exemples aléatoires. Avec le SGD, le résultat change un peu (moins de 10%) chaque cas de test car cet algorithme choisit aléatoirement des exemples dans chaque itération, donc, nous avons testé 10 fois chaque exemple et fait la moyenne pour comparer avec le résultat de la bibliothèque LibSVM.

Données	#Exemple	LIBSVM(%)	SGD(%)	LIBSVM(s)	SGD(s)	$\frac{SVM(s)}{SGD(s)}$
a1a	1,605	83.82	84.30	0.438	0.044	10
a2a	2,265	84.27	84.48	0.826	0.045	18
a3a	3,185	84.33	84.31	6.990	0.050	139
a4a	4,781	84.44	84.33	3.162	0.043	73
a5a	6,414	84.39	84.33	5.766	0.048	120
a6a	11,220	84.72	84.34	20.846	0.049	425
a7a	16,100	84.83	84.45	42.392	0.056	757
a8a	22,696	85.16	84.95	91.500	0.054	1,694
a9a	32,561	84.97	84.64	299.648	0.064	4,682

TABLE 4.2 – Comparaison entre LIBSVM et SGD-SVM

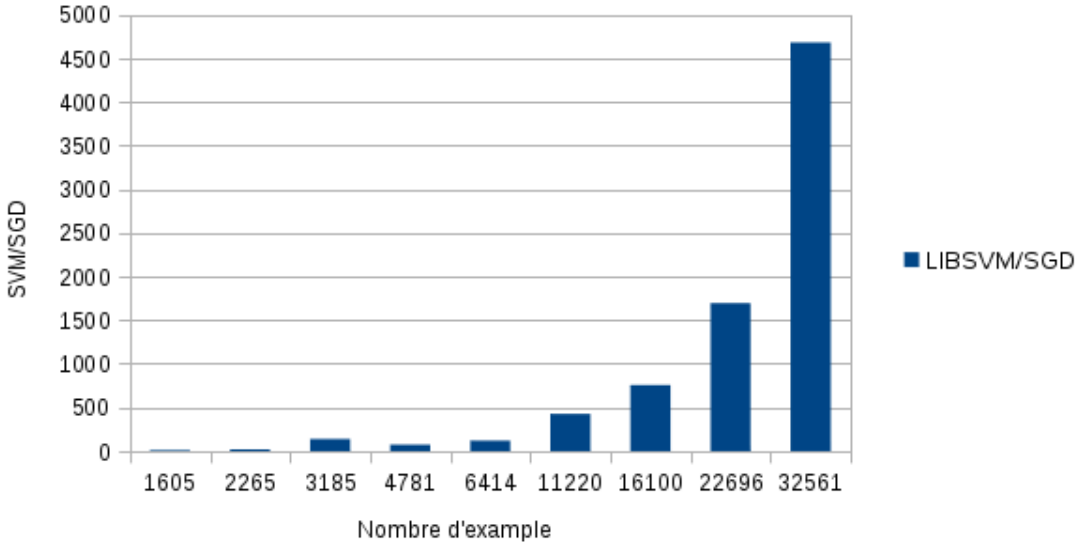


FIGURE 4.1 – Comparaison de la vitesse entre LibSVM et SGD binaire

Par rapport au table 4.2, nous trouvons que le taux de classification du SGD et de la LibSVM est presque pareil tandis que le temps d'apprendre est différent. La vitesse du SGD est très vite que la vitesse de la LibSVM. Cet avantage du SGD est plus clair quand le nombre d'exemples d'apprentissage augmente. Cette chiffre est démontrée dans la colonne 7 de table 4.2 ou aussi dans le figure 4.1.

Les données que nous avons utilisé ont 123 caractéristiques. Nous trouvons que quand le nombre d'exemple est de 1,605, l'algorithme SGD est plus vitesse que la LibSVM de 10 fois, mais quand le nombre d'exemple de 32,561, le SGD plus vitesse que la LibSVM 4,682 fois. Ce chiffre confirme que le SGD est beaucoup plus vitesse que la LibSVM, surtout pour les grandes données. Donc, l'algorithme SGD s'adapte mieux au problème de classification d'image dont la base de données est très grande. En raison de son avantage, nous voulons le développer pour qu'elle puisse résoudre le problème de classification multi-classes afin d'utiliser dans le domaine de classification d'images.

4.5 MC-SGD pour la classification multi-classes

- Protein : 3 classes, 17000 exemples, 357 caractéristiques
- Mnist : 10 classes, 60000 exemples, 780 caractéristiques
- SVM : function linéaire
- SGD : -iter 500 -k 100 -lambda 0.05

Données	SVM(%)	1-vs-all(%)	1-vs-1(%)	SVM(s)	1-vs-all(s)	1-vs-1(s)	$\frac{SVM(s)}{SGD(s)}$
Protein	68.23	68.41	69.10	551	0.20	0.19	**2755**
Mnist	86.92	86.46	89.71	2810	0.72	4.23	**3902.8**

TABLE 4.3 – Comparaison entre LIBSVM et MC-SGD parallèle

En générale, le résultat de classification du MC-SGD ne meilleure pas quand on compare avec la LibSVM. Par contre, le temps d'apprendre est beaucoup plus vite que la LibSVM pour le problème de classification multi-classes. La table 4.3 est une preuve. Pour le MC-SGD, nous trouvons que le résultat de classification de la version one-vs-one est un peu mieux que la version one-vs-all. Le temps d'apprentissage de deux options est presque pareil pour le problème de 3 classes. Par contre, pour le problème de 10 classes, la version one-vs-one est plus lent que la version one-vs-all 5.8 fois. Ce chiffre va augmenter très vite quand le nombre de classes augmente. Pendant ce stage, nous nous concentrons à développer un algorithme qui peut améliorer le temps du SVM, donc, nous trouvons que la version one-vs-all est un bon choix. Pour les tests suivants, nous ne faisons que la comparaison entre la version one-vs-all du MC-SGD avec la LibSVM.

4.6 MC-SGD pour la classification d'images

Comme nous avons parlé dans le processus, notre processus pour la classification d'images comprend 3 étapes principales : extraction des descripteurs avec le descripteur SIFT, construit le dictionnaire avec la méthode K-moyenne et l'apprentissage automatique pour la classification avec la méthode MC-SGD. Dans la partie précédente nous avons présenté le résultat de l'algorithme MC-SGD (l'étape 3, l'étape d'apprentissage automatique) avec la base de données existante. Maintenant, nous appliquons la méthode Sac de Mots pour préparer les entrées pour l'algorithme MC-SGD avec les bases d'images pour voir si la méthode MC-SGD s'adapte pour le problème de classification d'images.

- SVM : function linéaire
- SGD : -iter 1000 -k 10 -lambda 0.6

Données	SVM(%)	SGD(%)	SVM(s)	SGD(s)	$\frac{SVM(s)}{SGD(s)}$
Cal 101	61.52	65.12	2873	106.95	**26.9**
Cal 7 3D	91.52	88.3	113.4	0.81	**140**
ImgNet 3d	76.54	75.8	144	0.90	**160**
ImgNet	84.08	86.58	327	1.64	**199.4**

TABLE 4.4 – Comparaison entre LIBSVM et MC-SGD parallèle pour la classification d'images

Le tableau 4.4 montre que le pourcentage de classification de la LibSVM est presque égale au MC-SGD. Par contre, la LibSVM est plus lent que le MC-SGD de 26 à 199 fois. Nous rappelons que la LibSVM utilise l'option one-vs-one et le MC-SGD utilise l'option one-vs-all. Donc, si la base d'image augmente plus de catégories (le nombre de classes), l'algorithme MC-SGD sera plus vitesse si l'on compare avec l'algorithme de la LibSVM.

Pour être facile de faire la comparaison sur la vitesse entre notre algorithme et la LibSVM, nous vous montrons la graphique 4.2 (le taux de SVM/MC-SGD)

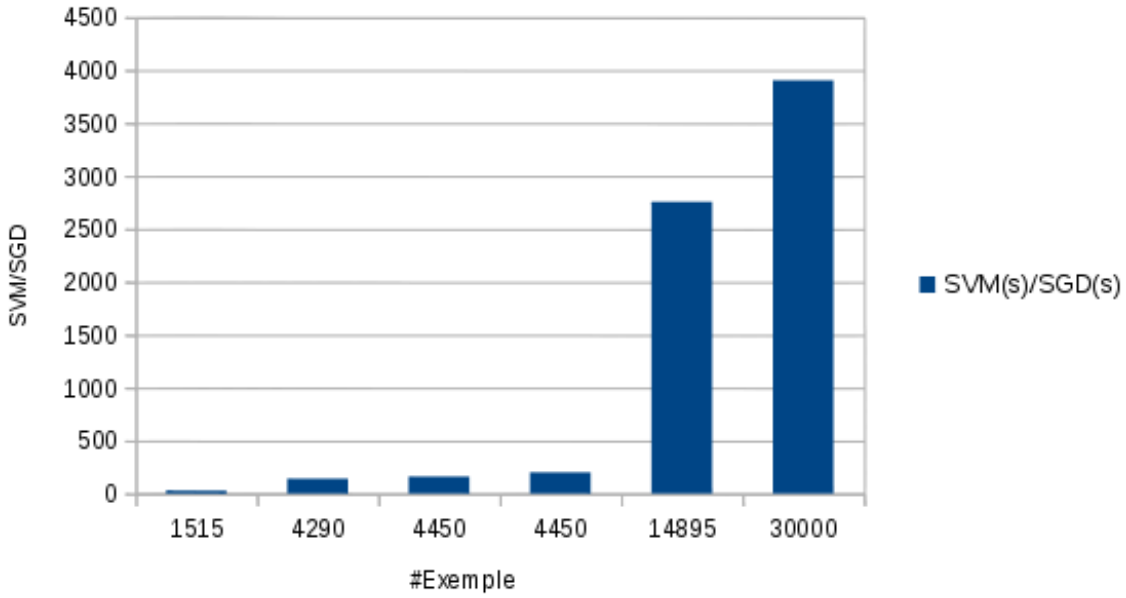


FIGURE 4.2 – Comparaison de SVM et MC-SGD (SVM/MC-SGD)

Non seulement plus vite que le SVM quand le nombre de classes augmente, mais aussi sur le nombre d'exemple d'apprentissage. La graphique ci-dessus peut montrer ce que nous expliquons. Le taux de temps de SVM/MC-SGD augmente selon le nombre d'exemple très claire dans cette graphique. Précisément, quand le nombre d'exemple de 1515, ce taux est de 26.9 mais ce taux est de 199.4 quand le nombre d'exemple est de 60000. La raisons est que notre algorithme n'a pas besoin de prendre tous les exemples pour l'apprentissage.

Conclusion et perspectives

Se basant sur la méthode SVM, une méthode très populaire dans le domaine d'apprentissage automatique, les auteurs ont développé Pegasos [2] utilisant le SGD binaire. En générale, la méthode Pegasos n'est pas meilleure que la méthode SVM sur la classification car le SGD est une version simple du SVM où l'on ne doit pas résoudre le problème de programme quadratique. Se basant sur SGD binaire, nous avons développé notre algorithme pour que SGD s'adapte bien au problème de multi-classes. Nous avons implémenté avec toutes les deux options possibles one-vs-one et one-vs-all. Pour le problème de classification de multi-classes avec one-vs-all, le SGD Pegasos trouve le problème de balancé. Pour cette raison, nous avons fait la échantillonnage et l'équilibra de la méthode SGD Pegasos. Dans cette recherche, nous ne faisons pas le point sur le résultat de classification, mais sur la vitesse de classification sur les bases d'images réelles. Donc, le MC-SGD s'adapte bien. Pour étudier notre algorithme, nous avons développé MC-SGD-Toy.

Bien que le MC-SGD a des avantages de la vitesse, cette méthode soit difficile de choisir des paramètres entrées tel que le nombre d'itération, lamda. Dans l'avenir, nous étudierons pour chercher des paramètres optimales de la méthode. Nous testerons aussi des bases d'images plus grandes tel que ImageNet. Nous développerons pour le SGD s'adapte aux autres domaines, tel que à la classification de vidéos.

Bibliographie

[1] David G. Lowe, *Object Recognition from Local Scale-Invariant Features,* In proceeding of the 7th International conference of computer vision, pages 1150 - 1157, Corfou, Grèce, 1999.

[2] Shalev-Shwartz, S.Singer, Y.Srebro, *Primal esti-mated sub-gradient solver for svm.*, Proceedings of the Twenty-Fourth International Conference Machine Learning. pp. 807–814, ACM (2007).

[3] *LIBSVM Data : Classification (Multi-class)*, `<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`

[4] PHAM Nguyen Khang, *Analyse factorielle des correspondances pour l'indexation et la recherche d'information dans une grande base de données d'image*, thèse IRISA, 2009.

[5] C.Schmid and R.Mohr, *Local Greyvalue Invariants for Image Retrieval*, IEEE Transactions on Pattern Analysis and Machine Intelligence,530–534, 19, 5 (1997).

[6] Chris Harris and Mike Stephens, *A combined corner and edge de detector*, in Alvey Vision Conf., 1988, pp. 147 - 151.

[7] Y.Dufournaud, C.Schmid and R.Horaud, *Matching Images with Different Resolutions*, in "International Conference on Computer Vision and Pattern Recognition (CVPR '00) 1 (2000) 612–618"

[8] Tony Lindeberg, *Feature Detection with Automatic Scale Selection,* Technical report ISRN KTH/NA/P–96/18–SE, May 1996, Revised August 1998. Int. J. of Computer Vision, vol 30, number 2, 1998. (In press).

[9] T-N DO, N-K PHAM, *Phan lop anh voi giai thuat giam gradient ngau nhien da lop,* Magasin Cantho university, 29 (2013).

[10] K.Mikolajczyk and C.Schmid, *Indexing based on scale invariant interest points*, in "Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on (Volume :1)", pages 525 - 531, 2001.

[11] David G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*, International journal of computer vision, 60(2), 91 - 110, 2004.

[12] S.Belongie, J.Malik and J.Puzicha, *Shape matching and object recognition using shape contexts*, in "Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume :24 , Issue : 4)", pages 509 - 522, 2002.

[13] Yan Ke and Rahul Sukthankar, *PCA-SIFT : A More Distinctive Representation for Local Image Descriptors*, in "Computer Vision and Pattern Recognition", 2004.

[14] K.Mikolajczyk and C. Schmid, *A performance evaluation of local descriptors*, in "Pattern Analysis and Machine Intelligence, IEEE Transactions on (Volume :27 , Issue : 10)", 2005.

[15] Marco Treiber, *An introduction to object recognition : selected algorithms for a wide variety of application*, ISBN 9781849962346, pages p. 147, 2010.

[16] J.MacQueen, *Some methods for classification and analysis of multivariate observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press Vol.1, pp. 281-297 (1967).

[17] J.Platt, *Sequential Minimal Optimization : A Fast Algorithm for Training Support Vector Machines*, Microsoft Research Technical Report MSR-TR-98-14 (1998).

[18] C.Chang and C.Lin, *LIBSVM – a library for support vector machines*, http ://www.csie.ntu.edu.tw/ cjlin/libsvm (2001).

[19] J.Weston, C.Watkins*Support vector machines for multi-class pattern recognition.*, Proceedings of the Seventh European Symposium on Artificial Neural Networks. pp. 219–224. (1999)

[20] V.Vapnik, *The Nature of Statistical Learning Theory.*, Springer, New York (1995).

[21] V.Vapnik, et S. Kotz *Estimation of Dependences Based on Empirical Data,* Springer Series in Statistics, 1982, ISBN 978-0387907338.

[22] U.Krebel, *Pairwise classification and support vector machines*, Support Vector Learning, Advances in Kernel Methods. pp.255–268. (1999).

[23] A.Bosch, A.Zisserman and X.Munoz *Scene classification via pLSA.*, Proceedings of the European Conference on Computer Vision, pp. 517–530 (2006).

[24] J.MacQueen *Some methods for classification and analysis of multivariate observations.*, Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press Vol.1, pp. 281-297 (1967).

[25] Nguyen-Khang Pham http ://www.cit.ctu.edu.vn/ pnkhang/index-en.html, Can Tho University

[26] *LIBSVM Data : Classification (Binary Class)*, `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

Table des figures

Liste des tableaux

Liste des algorithmes