Kernel Methods in machine learning
Data challenge

Charline Tessereau, *MSV* tessereau.charline@gmail.com

Philippe Peroumal, *MVA* Philippe.PEROUMAL@ensae-paristech.fr

Badr-Eddine Chérief-Abdellatif, *MVA* Badr.Eddine.CHERIEF.ABDELLATIF@ensae-paristech.fr

Méthodologie

Pour ce projet, nous avons réparti les tâches, chacun codait au départ de son côté, sur des algorithmes différents. Chacun a pu coder ses idées dans le langage qu'il souhaitait. Bien sur, nous communiquions afin de nous entraider si nous étions bloqués.

Les langages utilisés sont matlab et python.

Le code contient en tout 5 notebooks Python. Notamment, un pour chaque noyau utilisé (nommés "Final Kaggle MVA Kernel Linear", "Final Kaggle MVA Kernel Polynomial", "Final Kaggle MVA Kernel RBF"). Le notebook le moins détaillé est le notebook correspondant au noyau linéaire, car c'est celui qui donne les moins bons résultats. A l'inverse, le noyau RBF donne de très bons résultats et son notebook est donc très détaillé et utilisé pour la soumission finale. Les deux notebooks restants correspondent à notre décomposition en ondelettes (dans "ondelettes V1" et "ondelettes LOAD"). Il contient aussi 2 codes matlab ("Algo1 Comparaison Kernel" et "algo2 compa multiple kernels") mettant en oeuvre deux nouvelles méthodes d'utilisation des noyaux, et faisant appel à cinq fonctions externes ("multiple distance", "multiple k", "distance", "k", "selectNlignes").

Algorithmes utilisés

Deux algorithmes ressemblant à des k-mean

L'algorithme Algo1_Comparaison_kernels codé en matlab fonctionne de la manière suivante :
On sépare les données en sélectionnant aléatoirement N lignes (souvent N=3700 images utilisées pour l'entraînement et 1300 pour le test, pour respecter le rapport 2000/5000 de l'exercice) grâce à la fonction selectNlignes.

L'idée est de comparer les valeurs d'un noyau entre un élément et chaque classe, grâce à la fonction distance.m. On peut essayer différents noyaux en commentant ou décommentant dans la fonction k.m (linéaire, quadratique, exponentiel, exponentiel quadratique, tangente).

L'algorithme Algo2_Comparaison_multiple_kernels fait l'algorithme suivant pour différents kernels (linéaire, quadratique, exponentiel, exponentiel quadratique, tangente), et prend le vote majoritaire parmis tous les noyaux. Il s'agit d'une méthode classique d'aggrégation de modèles. Les codes correpondants sont distances_multiple_kernels.m et multiples_kernels.m.

SVM

Nous avons par la suite implémenté un algorithme SVM qui permet de classifier nos images. Tout d'abord, nous avons écrit, à l'aide des classes sous Python, des fonctions adaptées à un problème binaire. Afin de le généraliser à un problème de classification multi-classes comme c'est le cas ici, nous avons utilisé les méthodes One-Vs-One et One-Vs-Rest. La première approche consiste à construire 10*(10-1)/2 = 45 classificateurs (chacun correspondant à une paire de modalités) tandis que la seconde consiste à construire 10 classificateurs (chacun correspondant, pour chaque modalité, à regrouper toutes les autres comme en formant une seule). Nous avons procédé ainsi pour chaque noyau.

Nous avons considéré trois noyaux :

- Le noyau linéaire, qui ne contient aucun paramètre.
- Le noyau polynomial, qui contient deux paramètres, la constante et le degré.
- Le noyau RBF, qui contient le paramètre σ.

Notons qu'il faut ajouter à ces paramètres le C intrinsèque à la formulation duale du SVM. Nous avons fait varier ces paramètres entre 1 et 10 afin de sélectionner le modèle optimal.

Nous parvenions avec ces méthodes tout au plus à des scores compris entre 20% et 25%, selon les paramètres et le type de classificateur (One-Vs-One et One-Vs-Rest) que nous choisissions. Il s'agissait alors de trouver d'autres moyens d'augmenter significativement les résultats autrement qu'en optimisant simplement les paramètres parmi un nombre choisi.

Nous avons aussi essayé une réduction de dimension, en implémentant l'algorithme d'Analyse en Composante Principale (PCA et k-PCA) pour réduire la dimension des données tout en conservant les corrélations possibles entre les variables. Il s'agit de projeter notre base sur les vecteurs propres associés aux plus grandes valeurs propres de la matrice de variance (ou de Gram) de notre base de données, mais cela n'a pas donné de résultats fructueux. Cela peut s'expliquer par le fait que l'information contenue par les pixels est répartie dans de nombreuses dimensions. Toutefois, nous aurions peut-être pu obtenir des résultats plus satisfaisants si nous avions intégré à notre implémentation un réel critère de sélection du nombre de composantes à conserver non arbitraire.

Histograms of oriented gradients

L'idée pour augmenter les résultats était d'utiliser des méthodes de détection de forme et de contour. En calculant les gradients discrets en chaque pixel, il est possible de rendre compte des changements d'intensité. Ainsi, nous avons utilisé la méthode des "Histograms of Oriented Gradients "qui permet d'extraire des features de nos bases de données sur lesquels appliquer directement nos algorithmes de classification. Une brève description des paramètres choisis lors de notre implémentation de l' "Histograms of Oriented Gradients" est directement accessible sur les notebooks Python.

Cette méthode de feature extraction a donné des résultats très satisfaisant. Nous souhaitions tout de même améliorer à nouveau nos résultats, et c'est la raison pour laquelle nous nous sommes intéressés aux ondelettes.

Transformée en ondelettes

Nous avons opéré une transformation en ondelettes sur chaque niveau de couleur R,G,B. Nous n'avons pas eu recours à des librairies externes. Nous utilisé une base d'ondelettes polynomiales, la starlet, qui se prêtait à un codage simple. Nous gardons un niveau de détail. Ainsi nous avons obtenu des images avec des contours plus nets. Toutefois, cela n'a pas permis d'améliorer nos performances.

Résultats

Les deux algorithmes en matlab donnaient une bonne classification d'environ 0.20 maximum.

Durant l'élaboration de nos modèles, nous les évaluions en répartissant notre base Xtr dans les proportions Train/Validation 60%/40%. Nous avons finalement retenu la méthode SVM One-Vs-Rest pour le noyau RBF appliqué aux features obtenus par HOG, avec $C = 1$ et $\sigma = 2$.

Conclusion

Les algorithmes que nous avons utilisés ont donné des résultats satisfaisants. Nous pourrions toutefois les améliorer en choisissant des critères d'optimisation plus solides. Notamment, nous pourrions élargir le champ des possibles parmi les valeurs prises par nos différents paramètres. De même, il pourrait être intéressant de pouvoir faire varier la taille des blocs dans l'implémentation des HOG, ou encore le nombre d'orientations des histogrammes. De plus, le critère de performance que nous choisissons pourrait être amélioré en effectuant une validation croisée ou en adoptant un système d'agrégation ou de vote de classificateurs.