RDFIA TME 2-Bis
Classification d'images par SVM

Arthur Douillard Asya Grechka Alexandre Rame Matthieu Cord

21 octobre 2018

Comptes rendus à rendre

— Ce TP est un "*Pour aller plus loin*" pour ceux ayant tout fini la partie SIFT - Dico Visuel - BoW. Ce TP ne sera pas noté et aucun compte rendu sera attendu (sauf si vous le voulez, mais cela ne sera pas noté !)

Les données et une version numérique du sujet sont accessibles
à l'adresse `https://arthurdouillard.com/rdfia`

Introduction

Durant les précédents TP, nous avons mis en place une chaine de traitement nous permettant de transformer une image en une représentation numérique compacte adaptée à la classification des images. Ainsi, chaque image est représentée sous la forme d'un vecteur BoW décrivant en quelque sorte le taux de présence de $p = 1001$ "pattern-types" dans l'image, sous la forme d'un vecteur $\mathbf{x}_i \in \mathbb{R}^p$. Par ailleurs, on sait que chaque image appartient à une des 15 catégories de notre jeu de données. Durant ce TP, nous allons donc mettre en place classifieur d'image en utilisant le modèle SVM (Support Vector Machine).

Partie 1 – Rappels sur la classification et les SVM

Apprentissage. Un algorithme de classification automatique à pour but "d'apprendre" à classer des données, c'est à dire trouver une fonction $f_{\mathbf{w}}$ de paramètres $\mathbf{w}$ tel que $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{x})$ permet de produire la catégorie $\hat{\mathbf{y}}$ correcte compte tenu de l'entrée $\mathbf{x}$. Pour cela, on utilise un ensemble d'apprentissage $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1..N_{train}}$. Pour cela, un algorithme d'apprentissage nous permet de trouver les paramètres optimaux $\mathbf{w}^*$ qui minimisent une fonction de cout $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})\right] = \arg\min_{\mathbf{w}} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \left[\mathcal{L}(\mathbf{y}, f_{\mathbf{w}}(\mathbf{x}))\right] \tag{1}$$

SVM. Dans notre cas, $\mathbf{x}$ est la représentation BoW d'une image, $y = \{-1, 1\}$ indique si l'image appartient ou non à une classe donnée, f est un SVM donc un modèle linéaire $f(\mathbf{x}) = \mathbf{w}\mathbf{x} + b$, et la fonction de cout est celle d'un SVM, c'est-à-dire $\mathcal{L} = \max(0, 1 - y(\mathbf{w}\mathbf{x} + b)) + \lambda||\mathbf{w}||_2^2$ où λ est un *hyper-paramètre* du modèle de classification, c'est-à-dire un paramètre *sur l'optimisation/le modèle*, par opposition aux paramètres *optimisés* $\mathbf{w}$.

Cas multi-classe. Notez que parce qu'un SVM est un classifieur binaire qui determine si $\mathbf{x}$ appartient ou non à une seule classe, il nous est impossible de l'appliquer directement à notre problème multi-classes. À la place, nous allons donc appliquer une stratégie *one-vs-all* où on apprend un classifieur par classe, où chacun doit déterminer si l'entrée $\mathbf{x}$ appartient à chacune des classes du dataset. La prédiction finale consistera à choisir la classe avec le plus fort score prédit.

Évaluation. Notre modèle est appris sur un ensemble d'exemples limité que l'on appelle **jeu d'apprentissage**. Le modèle appris peut être appliqué à des exemples pour que l'on mesure sa précision (accuracy), c'est à dire le nombre d'exemples bien classés. On peut calculer cela sur l'ensemble d'apprentissage, cependant, il est possible que notre modèle ait simplement appris "par coeur" ces exemples (on appelle ce phénomène le **sur-apprentissage**) mais ne **généralise** pas à des exemples qui lui sont inconnus. Pour évaluer correctement la qualité de notre modèle, on garde un **jeu de test** que l'on ne montre pas au modèle pendant son apprentissage, et qui nous permet d'avoir une meilleure idée de la qualité du modèle. On pourra également garder un **jeu de validation** qui permettra de choisir les valeurs des hyper-paramètres, pour ne garder le jeu de test que pour le test final.

Partie 2 – Pratique

Nous allons donc désormais travailler sur les représentations BoW de nos images. Vous pouvez donc soit les calculer vous même avec le code réalisé lors des TP précédents, soit utiliser les données fournies dans le fichier `15_scenes_Xy.npz` qui est téléchargé au début du Colab.

Pour utiliser les représentations fournies, chargez le fichier avec la fonction `data = np.load("15_scenes_Xy.npz", "rb")`. Vous aurez alors `data["X"]` et `data["y"]`.

À faire

- Préparer / charger la matrice $\mathbf{X}$ et le vecteur $\mathbf{y}$ de features et labels
- Séparer vos données en des ensemble de *train* / *val* / *test* avec 70% / 10% / 20% des données. Utilisez la fonction `train_test_split` de scikit-learn.
- Apprendre un SVM (utilisez scikit-learn) sur vos données de train avec différentes valeurs de C ("équivalent" au λ ci-dessus). Commencez par $C = 1$ pour tester avant d'en apprendre d'autres.
- Evaluez vos SVM sur l'ensemble de validation pour trouver la valeur optimale de C.
- Evaluez votre meilleur SVM sur l'ensemble de test pour obtenir votre performance finale.
- Répétez les étapes précédentes avec d'autres paramètres du SVM. Par exemple le kernel (linear, rbg, etc.), ou la fonction de décision (un contre un, un contre tous).

Questions

Ces questions sont pour vous ; aucun rendu n'est attendu.

1. Discutez les résultats obtenus, tracez pour chaque hyperparamètre une courbe de performance en fonction de C, du kernel, et de la fonction de décision.
2. Quel est l'effet de chaque hyperparamètre ?
3. Pourquoi l'ensemble de validation est-il nécéssaire en plus de l'ensemble de test ?
4. Supposons que l'on souhaite utiliser ce classifieur sur de nouvelles images. À partir d'une image noir et blanc fournie, décrivez la chaine de traitement qui permet d'obtenir la prédiction du type de scène de l'image.