TP classification : régression logistique et softmax

Master 2 IM

14 décembre 2015

1 Introduction

Dans la première partie de ce TP, il s'agit d'implémenter une régression logistique. Puis, une régression softmax pour généraliser au cas multiclasse. Vous testerez votre algorithme softmax sur une tâche de reconnaissance de chiffres manuscrits issus du fameux corpus MNIST. Le langage de programmation est Matlab. Les fonctions nécessaires pour l'optimisation, le chargement des images sont fournies.

2 Régression logistique

Ici, on s'intéresse à la classification binaire d'étudiants en admis / non-admis en fonction de leurs notes obtenues à deux examens. Pour cette partie, le fichier principal est `logistiqueMain.m`. Il réalise toutes les étapes nécessaires, et vous n'avez a piori pas à le modifier.

Vous devez suivre étape par étape le déroulement du script principal et implémenter au fur et à mesure les fonctions suivantes :

- `sigmoid.m`
- `logisticCost.m`
- `logisticPredict.m`

Rappels La probabilité de prédiction de la classe positive est déterminée à l'aide de la fonction logistique, également appelée sigmoide :

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)} \tag{1}$$

La fonction de coût à minimiser et son gradient sont :

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \tag{2}$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{3}$$

3 Régression softmax

Pour cette partie, le fichier principal est `softmaxMain.m`. Il réalise toutes les étapes nécessaires, et vous n'avez a piori pas à le modifier.

Il commence par initialiser des constantes et variables nécessaires, charge les données, et affiche 12 images sélectionnées aléatoirement dans le corpus d'images de chiffres manuscrits appelé MNIST [1].

Il y a dix classes pour les dix chiffres possibles. Le chiffre 0 est mappé sur la classe numéro 10, pour faciliter l'implémentation du fait que les indices de tableaux commencent à 1 sous Matlab.

Vous allez devoir implémenter les fonctions suivantes :

— `softmaxCost.m`

— `softmaxPredict.m`

3.1 Coût et gradient

Dans un premier temps, il s'agit d'implémenter la fonction `softmaxCost.m` pour qu'elle retourne le coût (`cout`) et le gradient (`grad`). Suivez les étapes décrites ci-dessous.

3.1.1 Prédictions

La première étape consiste à implémenter directement dans la fonction `softmaxCost.m` les prédictions trouvées avec les paramètres `thetagrad` sur les données `data`, passés en argument.

Pour chaque vecteur de données de test $x^{(i)}$, il s'agit de calculer un vecteur de probabilités d'obtenir l'une des k classes (avec $k = 10$). Ces probabilités suivent l'équation suivante :

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix} \tag{4}$$

Vous devez vectoriser votre implémentation, c'est-à-dire utiliser des opérations matricielles pour éviter d'utiliser des boucles `for`. Ainsi, calculez les prédictions sous la forme d'une unique matrice de dimension $k \times m$ avec k le nombre classes, et m le nombre de vecteurs de test de `data`.

Avant de prendre l'exponentielle, veuillez à soustraire le max de chaque colonne à chaque élément par colonne. Ceci à l'aide de cette instruction, pour une matrice quelconque M :

```
M = bsxfun(@minus, M, max(M, [], 1));
```

Cela permet d'eviter des overflows potentiels dûs à l'exponentielle.

Pour diviser tous les élements des colonnes par leur somme en colonne, vous pouvez utiliser cette instruction :

```
M = bsxfun(@rdivide, M, sum(M));
```

1. `http://yann.lecun.com/exdb/mnist/`

3.1.2 Coût

Implémentez ensuite le coût $J(\theta)$, donné par la formule, où $1\{\}$ est la fonction indicatrice :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} 1\left\{y^{(i)} = j\right\} \log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}} \tag{5}$$

Vous pouvez utiliser la matrice `groundTruth` pour éviter l'usage de boucles `for`.

3.1.3 Gradient

Implémentez le gradient `thetagrad` :

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}\left(1\{y^{(i)} = j\} - p(y^{(i)} = j|x^{(i)};\theta)\right)\right] \tag{6}$$

De nouveau, vous pouvez utiliser la matrice `groundTruth`.

3.1.4 Vérification du gradient

Exécutez softmaxMain avec le booléen DEBUG à true, jusqu'à l'étape 3 incluse pour comparer vos valeurs de gradient à celles calculées par une fonction qui vous est fournie, qui approxime le gradient numériquement, comme expliqué en cours.

3.2 Apprentissage des paramètres

Une fois que vous avez vérifié vos valeurs de gradients, vous pouvez lancer l'apprentissage des paramètres qui fait appel à la fonction `softmaxTrain`. Attention, mettez le booléen DEBUG à false pour ne pas procéder à la vérification du gradient qui est coûteuse en temps de calcul. Cette fonction utilise une fonction qui vous est fournie, appelée minFunc, qui offre différents algorithmes d'optimisation dont L-BFGS qui est utilisé dans ce TP.

3.3 Test

Une fois que l'apprentissage est terminé, vous pouvez tester votre modèle sur le sous-corpus de test de MNIST qui contient 10k images différentes de celles du corpus d'apprentissage. Vous devez implémenter la fonction `softmaxPredict.m`, en reprenant le code que vous avez produit dans la fonction `softmaxCost.m`.

Quel taux de reconnaissance obtenez-vous ?

3.4 Régularisation

Modifiez la fonction `softmaxCost.m` pour ajouter un terme de régularisation quadratique.

Entraînez un nouveau modèle en utilisant un coefficient de régularisation de `1e-4`.

Quel nouveau taux de reconnaissance obtenez-vous ?