Projet Final
Reconnaissance d'images

Professeur Guillaume Wisniewski

WANG Yuxiang
ZHANG Zhao
ZHOU Peikun
Le 5 Avril, 2016

Préface

Ce document a pour but de décrire le déroulement de notre projet Reconnaissance d'images. C'est le résultat du travail que nous avons obtenu.
Ce rapport contient l'ensemble des éléments du projet. Du point de vue technique, tout d'abord, nous commençons par nous familiariser avec le format de la base CIFAR-10, et nous répondrons à quelques questions, c'est la section 1.
Dans la section 2, nous traiterons la tâche en utilisant une méthode de classification qui est considérée comme la <valeur> de chaque pixel. La méthode est l'algorithme des k-moyennes. C'est l'approche naïve, nous avons résolu certains de ces problèmes. Avec l'algorithme des k-moyennes, nous avons obtenu le taux de précision 27,74%.
Enfin, nous avons implémenté et testé l'approche proposée SVM. En modifiant les paramètres et en modifiant les paquets de données, Nous apprenons la formation des données, ensuite nous testons les données pour les tests. Nous obtenons des résultats différents, et nous les présentons dans un tableau.

Le base de données Cifar-10

Nous utiliserons la base de données CIFAR-10 qui regroupe des images représentants 10 types d'objets différents. Cette base est disponible à l'url http://www.cs.toronto.edu/~kriz/cifar.html

Dans un premier rang, nous commençons par nous familiariser avec le format de la base. L'archive contient les fichiers data_batch_1, data_batch_2, ..., data_batch_5, Chacun de ces fichiers est un Python pickled objet qui est produit avec cPickle.

```
def unpickle(file):
    import cPickle
    fo = open(file, 'rb')
    dict = cPickle.load(fo)
    fo.close()
    return dict


#data, labels,batch_label, filenames


images =
unpickle('./data_batch_1')#.values()
[0]
data = images['data']
#print data
for key in images:
   print key, images[key];
 # print
 images.values()[0]
 # values =
 images.values()
```

Puis, nous savons que chacun des fichiers de commandes contient un dictionnaire contenant les éléments suivants :

Data : un 10000 x 3072 numpy array de uint8s. Chaque ligne du tableau stocke une image couleur 32x32, les 1024 premières entrées contiennent les valeurs du canal rouge, les suivantes 1024 les vertes, et les finale 1024 les bleues. L'image est stockée dans l'ordre des lignes-major, de sorte que les 32 premières entrées du tableau sont les valeurs de canal rouge de la première rangée de l'image.
Labels : une liste de 10000 numéros dans 0-9. Le nombre d'index i indique l'étiquette de l'image i dans les données du tableau.
Chacun de ces fichiers est formulé comme suit :

```
<1 x label><3072 x pixel>
...
<1 x label><3072 x pixel>
```

<1 x label> est le nom de Label d’image, il est un nombre dans l’intervalle 1-9, après les pixels <3072 x pixel> sont les données d’images en format de RGB, les premiers 1024 octets sont les valeurs de rouge, les suivants 1024 le vert, et les finals 1024 le bleu. Les valeurs sont stockées dans l'ordre des lignes-major, de sorte que les 32 premiers octets sont les valeurs de canal rouge de la première rangée de l'image.
Chaque fichier contient 10000 ces 3073 octets "lignes" d'images, bien qu'il n'y ait rien délimitant les rangées. Par conséquent, chaque fichier doit être exactement 30.730.000 octets.

L’algorithme des k-moyennes

K-moyennes est une méthode de partitionnement de données et un problème d’optimisation combinatoire. Étant donnés des points et un entier k, le problème est de diviser les points en k partitions, souvent appelés clusters, de façon à minimiser une certaine fonction. Nous considérons le centre le plus proche d’un point comme son cluster, la fonction cherche à minimiser la norme.

Il existe une heuristique classique pour ce problème, souvent appelée méthodes des k-moyennes, utilisée pour la plupart des applications. Le problème est aussi étudié comme un problème d’approximation.

Etant donné un ensemble de points (x1, x2, …, xn), on cherche à partitionner les n points dans k ensembles S = { S1, S2 , …, Sn }(k ≤ n) en minimisante la distance entre les points à l’intérieur de chaque partition.

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

Ou μi est la moyenne des points dans Si.

Classifier les images sur notre idée de la façon naïve :

K-moyennes est une méthode du partitionnement des données. Nous avons utilisé K-moyennes avec les labels. Etant donnés des données en K labels, le problème est de diviser des données en K partitions et de calculer les domaines pour chaque label. Après, s’il y a une autre donnée à tester, nous pouvons calculer le domaine qu’il est tombé.

En général, en méthode de K-moyennes, nous pensons que les deux points en distance les plus proches sont les deux les plus semblables.

Dans notre projet, d’abord, nous commençons par l’étape de l’apprentissage. Nous avons fait l’extrait des données en même label et calculé leur moyenne et on suppose que la moyenne soit le centre du domaine de ce label. Dans

notre projet, nous avons 10 labels et nous avons donc aussi 10 centres. Puis, nous testons des données. Maintenant, nous avons 10000 images, par la définition de K-moyennes, nous cherchons à partitionner ces 10000 images en 10 ensembles et en minimisant la distance entre les points à l'intérieur de chaque partition :

$$label = argmin \|x - u\|^2$$

Donc, pour notre projet, nous commençons par le traitement de la donnée. D'abord, nous avons fait de l'extrait des images qui sont en même label et les mis dans un ensemble. Comme la taille des données est très grande, nous voulons chercher des méthodes pour gagner le temps. En utilisant les fonction de np.means(p,axis =0) , nous pouvons calculer directement les centres des données en forme de la matrice. Après, en même raison, nous pouvons calculer les distance en utilisant la fonction np.linalg.norm(distance, axis=1) et rendre l'index de la valeur la plus petite comme son label en utilisant la fonction np.argmin(distance). Donc, pour chaque image à tester, nous supposons qu'il soit plus semblable aux images qui sont plus proches. Avec cette façon, chaque image peut être partitionnée à un ensemble avec un label.

Dans notre travail, après tester les images, il obtient un résultat avec un taux de précision 27.74%. Cela est meilleur que 10%(le taux original), mais pas assez bon pour traiter une image. Nous pensons que la façon de déterminer le centre d'un ensemble n'est pas assez précise et nous ne sommes pas sûrs que la distance puisse bien présenter la similarité. De plus, le temps calculé est peut-être exponentiel selon le nombre d'images. Donc, nous voulons chercher une autre méthode pour traiter ces images.

Réponse des questions

L’approche naïve

Une manière naïve de traiter la tâche est d’utiliser une méthode de classification considérant comme une caractéristique directe (la <valeur> de chaque pixel).

Comment représenter les pixels ?

Les pixels sont dans une liste d’une dimension avec une longueur de 3024. Elle est composée par trois parties : R(red) = data [:1024], G(green) = data[1024:2048] B(blue) = [2048:]. Chaque RGB est dans une matrix de 32 * 32. Les trois couleurs font la composition d'une image.

Par rapport à la composition des images, nous avons choisi 10 centres qui viennent de batch1-5(comme nous avons écrit avant). Ensuite nous traitons les pixels juste comme distances et calcule les plus proches entres les centres.

Quelle sont les performances obtenues ?

Nous avons obtenu le taux de précision 27,74% en 1 seconde. Ce n’est pas très haut mais assez vite, tenant compte d'utiliser 50.000 images.

Quelles sont les classes qui sont bien reconnues ?

Selon notre programme et le résultat des images. La première classe et la septième sont bien reconnues.

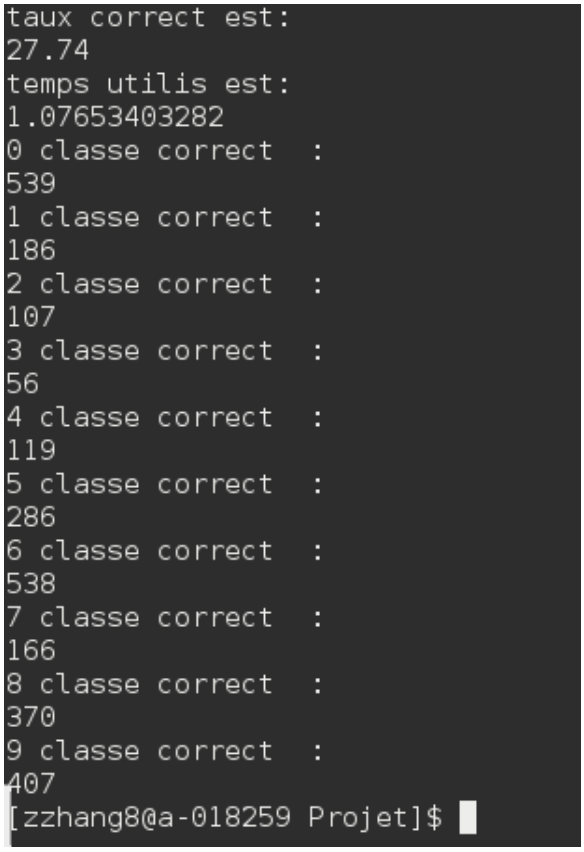


Figure 1

Quelles sont celles qui sont systématiquement mal reconnues ? Pourquoi ?

Dans notre programme, celle est la troisième. Car elle est proche de plusieurs centres. Et la distance en la distance euclidienne est très proche.

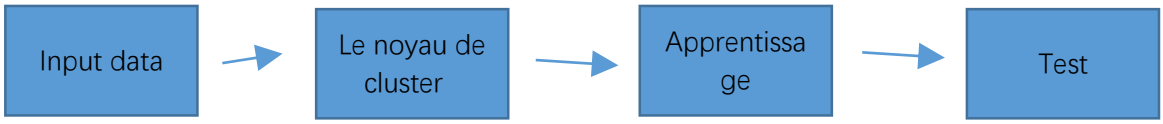
Peut-on utiliser ce partitionnement pour classer les objets ? Si oui, quelle performance obtient-on ?

Oui. Nous séparons 10 centres dans la programmation. K-moyenne n'est que pour partitionner. Ce n'est pas bien en comparant avec la mienne. Nous avons utilisé le K- moyenne. Elle utilise plus beaucoup de temps, et le taux correct est juste 18.04%(le cas meilleur).

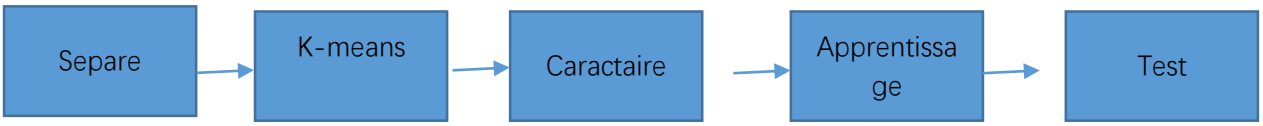
L’approche proposé

Nous allons maintenant mettre en œuvre la méthode SVM avec l’algorithme K-moyenne.

Support vector machines (SVMs) est un ensemble de méthodes d'apprentissage supervisé utilisés pour la détection de classification, la régression et les valeurs aberrantes. Le professeur a introduit cette méthode dans le dernier TP.
Nous avons introduit une amélioration d’algorithme K-moyenne à la section 2. Dans cette partie, nous traitons d’abord des données, nous obtenons les données d’images différentes, par exemple, nous appelons que chaque image est divisée en quatre ou seize. Puis nous calculons le noyau des clusters par k-moyenne. Nous attribuons à chaque image de vecteur caractéristique, ensuite, nous utilisons les résultats d’avant par SVM pour l’apprentissage, nous obtenons donc la répartition des vecteurs caractéristiques correspondant aux différents tags. En fin, nous utilisons la répartition pour tester les données et obtenons le taux de précision.



Nous définissons les différentes fonctions suivantes, les fonctions Séparées, K-means, Caractère, Apprentissage et Test. Le traitement est aussi comme suit:



Fonction Séparée

Le données format est 3072*10000/batch

Chaque image est une matrice de taille 1024, chaque colonne représentes R, G, B.

a. Divise en quatre, chaque de taille 16*16(1024 -> 4*16*16)
b. Divise en seize, chaque de taille 8*8(1024 -> 16*8*8)

```
Begin
Separe:
        Input a batch of image (data)
        Initialise all the variables l, new_list
        If image is in data, and data still has elements, then
                r<- image[0:1024]
                g <- image[1024:2048]
                b <- image[2048:3072]
        Get l-> a matrix of r,g,b with dimension 32*32*3
        If i is in I, and I still has elements, then
                        New_list get the pixels for a new little image with 16*16*3(a)
or 16*8*8(b)
return new_list
End
```

Fonction K-means

D'abord, nous obtenons K barycentres(centroids) par Random, puis en utilisant les k centroids pour calculer le noyau de cluster.

```
Begin
def Dis_eu(vector1, vector2):
   return the distance eclide between two vector
End

Begin
def returnCenter(dataSet,k):
       return center picked in random
End

Begin
def k_means(dataSet, k, distMeas=Dis_eu, createCent=returnCneter):
   Initialisation:
   m <- get the element number of dataSet
   Centroids <- get with fonction returnCenter
```

```
    clusterChanged <- True
    For each element in dataSet, gave the number of the nearest center number
to the element
        Update all the centers with the elements belong to them
        If centers don’t change, stop the boucle.
Return center
End
```

Fonction Caractère

Avec les image en style a ou b et le noyau, nous attribuons à chaque image de vecteur caractéristique. Si l’image a-i correspond à son noyau-j d’image, label[i][j] = 1, sinon, label[i][j] = 0. Dans cette fonction, nous utilisons deux façons, la première est d’utiliser 10000 images pour calculer les noyaux et les données d’apprentissage, l’autre est d’utiliser 20000 images.

Fonction Apprentissage

En utilisant SVM pour comparer les données d’apprentissage et ses originaux tags, puis nous les passons à SVM. Donc nous obtenons la répartition des vecteurs caractéristiques correspondants aux différents tags. Nous prédisons les vecteurs de caractéristiques d'image cible.
Ici, nous utilisons le lib : *sklearn* pour réduire le temps de calcul.

Test & Analyzer

Nous avons testé plusieurs groupes de données,

a. En séparant chaque image en 4 matrices de taille 16*16, nous utilisons de différentes tailles de données pour l'apprentissage, puis nous comparons les taux, les résultats

Nombre de centre	Taux de précision d'apprentissage 10000 images	Taux de précision d'apprentissage 20000 images	Taux de précision d'apprentissage 1000 images
20	27.15	28.12	19.31
30	28.54	29.88	26.31
40	28.58	32.45	-
50	28.01	30.04	-
60	24.82	29.07	-
70	21.19	29.81	-
100	19.79	25.52	13.19

Tableau 1

Dans ce tableau, vous pouvez voir qu'avec les augmentations de nombre de données, SVM et K- moyenne peuvent apprendre mieux qu'avant. Quand nous changeons plus de nombre de données, le taux devient haut.

b. Nous changeons le fonçons de diviser les images, nous divisons les images en 4 matrices, la première façon est de sélectionner les quatre coins, la deuxième façon est de sélectionner tous les 4 parties (matrices)
Nous comparons ces 2 groupes de données

Nombre de centre	Taux de précision d'apprentissage 10000 images en 4*10*10	Taux de précision d'apprentissage 10000 images en 4*16*16
20	19.47	27.15
30	23.91	28.54

Tableau 2

Selon tableau 2, vous pouvez voir que la façon de séparer les images a une grande influence sur les résultats. En théorie, si nous séparons assez beaucoup de caractères dans une zone, c'est mieux. Dans cette partie, la réalité conforme à la théorie.

c. Nous changeons la façon de diviser les images, et nous les divisons en 16 parties(matrices), ainsi chaque matrice de taille 8*8. Comparer les résultats avec les résultats de b.

Nombre de centre	Taux de précision d'apprentissage 10000 images en 4*10*10	Taux de précision d'apprentissage 10000 images en en 16*8*8
20	27.15	28.77
30	28.54	28.12
40	28.58	28.11
50	28.01	25.25
60	24.82	-
70	21.19	-
100	19.79	15.19

Tableau 3

En théorie, le taux de précision d'apprentissage 10000 images en en 16*8*8 faut être plus haut que le taux de précision d'apprentissage 10000 images en 4*10*10. Mais dans nos données le résultat est inverse. Nous pensons

qu'en cas d'apprendre trop de l'information, il ne peut pas bien obtenir le résultat correct.

d. Comparer les résultats de la façon naïve avec les résultats de propose (SVM+K- moyenne). En détail, nous obtenons les nombres de test_batch en chaque cluster.

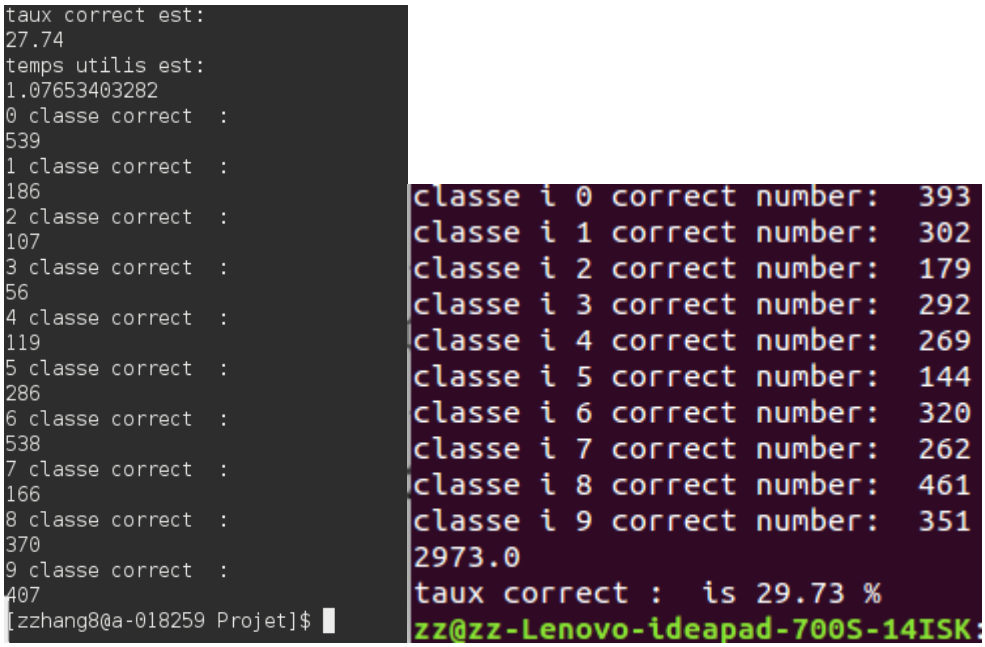


Figure 2.

e. Figure 2 est utilisé 30 centres, 10.00 pour apprendre, chaque image est séparée en 4 parties.

f. Les deux façons utilisent distance euclidienne. Nous comparons figure1 et figure 2. Vous pouvez voir que : avec les deux façons, pour chaque cluster, il y a trop de différences. Donc nous pouvons dire que les caractères que nous obtenons sont très différentes. Si nous séparons les image plus petite, nous pouvons avoir les caractères plus facile à distinguer. Donc, la façon proposée est beaucoup mieux que la façon naïve (dans notre cas, même si nous avons amélioré la naïve).

Conclusion

Nous avons fini notre projet sous la direction de monsieur Wisniewski, et nous avons obtenu nos résultats par l'algorithme naïve et l'algorithme SVM. Nous avons aussi obtenu le taux de précision plus de 30%(le meilleur est de 32.45%). A nos avis, l'algorithme proposé est meilleur que l'algorithme naïve. Mais nous pensons que nous devons faire plus d'efforts à trouver une meilleure méthode qui rend un meilleur taux de précision.

En détail, pour apprendre, si nous avons la taille de données plus grande, nous pouvons obtenir un meilleur résultat.

Pour aller plus loin, nous voulons essayer d'apprendre davantage d'algorithmes. De plus, par exemple, en utilisant la façon de *whitening*, peut-être nous pouvons avoir un meilleur résultat.