Proposition de projet - Vision par ordinateur

Projet: Vision – Equipe Regard
Membres de l'équipe : PETIT Marc-Alain <marc-alain.petit@u-psud.fr>, BEN BRAHIM Isra <isra.ben-brahim@u-psud.fr>, BARTHET Virgile <virgile.barthet@u-psud.fr>, DOUADY Paris <paris.douady@u-psud.fr>, BOURDON Theo <theo.bourdon@u-psud.fr>, RAKOTO Cedric <cedric.rakoto-andriantsilavo@u-psud.fr>.
URL du challenge : https://codalab.lri.fr/competitions/111
URL du repo Github : https://github.com/RegardL2MP/Regard

Introduction:

Objectif du projet

Être capable de reconnaître divers obstacles pouvant se dresser sur la route d'une voiture autonome, comme des animaux ou d'autres véhicules, afin de pouvoir éviter de potentiels accidents en adaptant la réponse de la voiture à l'entité reconnue. Ce challenge fera appel à des algorithmes de Machine Learning.

Données et description du problème

Les données ici sont le fruit d'un réseau convolutionnel profond sur le dataset "CIFAR-10" ce qui enlève toute possibilité d'interprétation des features qui sont au nombres de 256. Les labels en revanche sont plus pertinents car ils correspondent à des entités de la vie courante. Le problème est donc un problème de classification classique.

Description des classes:

1) Visualisation : *PETIT Marc-Alain et BEN BRAHIM Isra*

Nous nous chargerons dans cette partie de choisir la manière dont nous allons représenter graphiquement les données. Ce choix est important car il convergera vers le type de représentation qui facilite le plus l'accès à nos données et leur manipulation, tout en s'assurant que celles-ci soient bien regroupées. De plus, ce choix prendra en compte le fait que le type choisi devra permettre de faciliter le plus possible la compréhension de la façon dont les données seront représentées avant et après les phases d'apprentissage.
Notre modèle va prédire à quelle classe correspond une donnée, où chaque donnée sera un vecteur à 256 dimensions, et à partir des exemples qu'on lui a donné il va apprendre à transformer ce vecteur en classe. Par exemple (1, 0, 0.3, 2.1, ...) -> "grenouille" et (0, -2.12, 2.1, 3.91, ...) -> "avion".

Nous allons donc pour cela représenter une matrice de confusion et en évaluer les résultats afin de déterminer si l'intelligence artificielle renvoie un résultat satisfaisant, soit une précision moyenne par classe (BAC) autour de 86%. Pour représenter au mieux les résultats obtenus, nous allons utiliser un scatterplot, afin de voir au mieux les variances entre les résultats obtenus.

2) Preprocessing : *BOURDON Théo et BARTHET Virgile*

Le « preprocessing » consiste à sélectionner, trier, nettoyer les données afin de les rendre facilement utilisables par le classifieur et d'améliorer ses performances ainsi que sa vitesse d'apprentissage. Nous allons procéder ici en 3 étapes :

La première consistera à bien choisir les données que nous utiliserons, supprimer certains labels superflus dans la base de données (une voiture a peu de chances de rencontrer un bateau sur sa route par exemple).

La deuxième étape consistera à nettoyer la base de données, en retirant ou réparant des entrées incomplètes, et enfin à tenter de réduire la taille du dataset en se servant uniquement d'un échantillon représentatif au lieu du dataset complet afin de réduire le temps d'apprentissage du classifieur.

La troisième et dernière étape sera de transformer les données, en les normalisant, en sélectionnant certaines features à retirer via une ACP, et en cherchant des features redondantes via une "feature agglomeration".

Le squelette de la classe zPreProcessor peut être trouvé sur le repo github du groupe https://github.com/RegardL2MP/Regard

3) Modèle prédictif : *DOUADY Pâris et RAKOTO Cédric*

Notre binôme devra former un modèle de prédiction parmi la multitude de choix que propose Scikit-Learn, puis il faudra tuner les hyper-paramètres du modèle afin de battre la méthode automatique de AUTO-SKLEARN en trouvant des méthodes qui correspondent aux données choisies. Le travail effectué par l'équipe de visualisation et de pre-processing sera probablement d'une grande aide dans toutes ces étapes.
Des petites expériences avec scikit-learn ont été réalisées pour estimer différentes approches en utilisant la métrique BAC (Moyenne de la précision par classe) sur les données de validation. Un Gaussian Naive Bayes avec les paramètres par défaut atteint à peu près 85.7%, une random forest 82.5% et un perceptron simple couche autour de 86%. Au semestre dernier nous devions pour la matière "Vie Artificielle" coder un perceptron multi-couches, et en utilisant la méthode de régularisation Dropout, la normalisation par batch et l'optimiseur RMSProp sur un perceptron à 3 couches de 256 neurones nous avons réussi à obtenir 88.60%. Cela semble être l'approche la plus prometteuse bien que d'autres méthodes restent à vérifier.

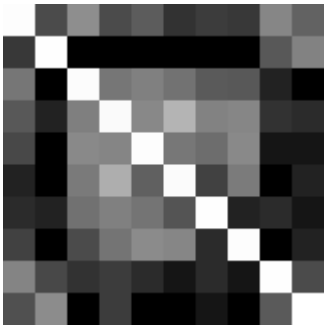
Algorithme en pseudo-code de l'interface graphique:

```
import datas //Importe les données
import matpotlib.pylab as plt  //Importe la bibliothèque graphique
import numpy as np  //Importe la bibliothèque de calculs mathématiques et scientifiques

//Vu notre modèle, on va effectuer un tracé en points (scaterplot), pour représenter les données.

Fonction lireDonnées{
        Importer Données
        Lire Données
        Recharger Données
}

//Dessin du Scaterplot

DessinnerPoints(){
        X_test = DonnéesUtilesConvertiesEnMatrice[ :,:-1]
        Y_test =  DonnéesUtilesConvertiesEnMatrice[:,-1]
        NormalisationMatriceX = Normaliser(X_test)
        NormalisationMatriceY = Normaliser(Y_test)
        NomAxeAbscisse = « NOM »
        NomAxeOrdonnées = « NOM »
        EchelleX0 = -30
        EchelleXMax = 30
        EchelleY0 = -30
        EchelleYMax = 30
        Tant que EchelleX0 < EchelleXMax{
                Dessin EchelleX0
                EchelleX0+10
        }
        Tant que EchelleY0 < EchelleYMax{
                Dessin EchelleY0
                EchelleY0+10
        }
        Dessin()
}

//Représentation de la matrice d'erreurs (1)

from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KneighborsClassifier
from sklearn.metrics import confusion_matrix
```

```
X_train, X_test, y_train, y_test = train_test_split(cifar_10.data,
cifar_10train.solution,test_size=1.0)
knn = KneighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_neighbors=5, p=2, weights='uniform')
y_pred = knn.predict(X_test)
confusion_matrix(y_test, y_pred)
knn.score(X_test, y_test)
```

Matrice de confusion pour un modèle simple:

Voici la matrice de confusion pour le perceptron 3 couches (les couleurs sont sur une échelle logarithmique et non linéaire pour une meilleure compréhension) :



airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

On peut voir que les classes “automobile”, “ship” et “truck” sont les plus simples à distinguer du reste, et que le couple (“cat”, “dog”) a la confusion la plus élevée.

Sources

Pour comprendre le fonctionnement des bibliothèques graphiques

http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx/notebooks/td1a_cenonce_session_12.html

https://python.developpez.com/tutoriels/graphique-2d/matplotlib/

http://pandas.pydata.org/pandas-docs/stable/visualization.html

(1) http://blog.xebia.fr/2015/04/29/les-outils-de-la-data-science-python-data-tools/

http://www.apnorton.com/blog/2016/12/19/Visualizing-Multidimensional-Data-in-Python/