Vision

Groupe Image

Membres du groupe

- Mariam Barhoumi mariam.barhoumi@u-psud.fr
- Leila-Yasmine Nedjar leila.nedjar@u-psud.fr
- Zakaria-Abderrahmen Difallah zakaria-abderrahmen.difallah@u-psud.fr
- Atte Torri atte.torri@u-psud.fr
- Matthieu Robeyns matthieu.robeyns@u-psud.fr
- *Jean-Marc Fares* jean-marc.fares@u-psud.fr

Page du challenge Vision
GitHub repository

Introduction

Contexte et motivation

L'actualité récente témoigne d'une part, de la faisabilité d'un système de conduite autonome sur un véhicule automobile [1-5], ainsi que d'un interêt non négligeable pour ce genre d'assistance à la conduite [Réf. nécessaire].
Afin de pouvoir envisager la réalisation d'un tel système, on se doit en premier lieu de pouvoir reconnaitre les différents obstacles qui seront présents sur le chemin du véhicule. Les données sont fournies par un capteur, généralement (comme pour [2,5]), il s'agît d'une caméra.

Problematique et Données

Lors de la conduite, un agent humain détecte et trie les stimulis (obstacles, panneaux, sons …) qu'il percoît pour les transformer en actions (accélération, freinage, changement de file, insertion …). Chacune des étapes est susceptible d'être acquise par un algorithme d'apprentissage [1, 2, 4]. Pour notre part, on se concentre sur l'interprétation des stimulis fournis par des capteurs, ici une caméra plus précisément. Nous nous donnons un jeu de données composé d'images représentant des objets ou obstacles pouvant être rencontrés sur une route : CIFAR10 [6]. Ce dernier se compose de 60'000 images colorées carrées de 32 pixels de côté réparties en 10 classes, soit 6'000 images par classes. Il comprend des images représentant des avions, des voitures, des oiseaux, des chats, des cerfs, des chiens, des grenouilles, des chevaux, des bateaux et des camions. Le créateur nous fournit ces données dans des fichiers objet `cPickle` archivés. Pour pouvoir exploiter les données, nous utilisons la fonction `unpickle` proposée par l'auteur [Annexe 3].

Axes de travail

Visualisation

La visualisation des données sert à mieux représenter les données brutes, pour permettre une lisibilité facile des données et voir plus clairement comment les données se séparent.

Plusieurs méthodes de visualisation existent, mais pour ce problème il est important d'utiliser une méthode qui réduit les dimensions, tout en conservant les structures complexes des données. Une première approche pourrait-être à l'aide d'une méthode linéaire tel que le PCA, qui consiste à créer de nouveaux axes avec de nouvelles variables, qui sont le produit des colonnes des matrices. Cependant, cette méthode est très pratique pour peu de données, or ce n'est pas notre cas. C'est pour cela que les méthodes non-linéaires utiles pour la visualisation des données dans notre cas sont le t-SNE, le Sammon Mapping et l'Isomap.

La méthode t-SNE permet de réduire les dimensions en créant des nuages de points en respectant les distances entre chaque point dans le

repère originel. Le problème de cette méthode est que sur une très grande dataset sa perpléxité (au sens de l'entropie de Sammon) peut atteindre des valeurs tel que 50. L'autre problème est que cette méthode utilise des formules quadratiques qui vont malheureusement écraser les petites variations de distances.

Le Sammon Mapping est une autre méthode non-linéaire, qui utilise les mêmes techniques que celles vues précédement mais elle minimise aussi les erreurs dites de Sammon en utilisant la descente de gradient. Le problème est que la descente de gradient est une méthode assez lente, car elle demande beaucoup d'itérations, et si les conditions initiales sont mal choisies, alors la méthohde devient obsolète.

L'Isomap est une méthode assez similaire aux deux autres, c'est-à-dire qu'elle va calculer les voisins de chaque point puis construire un graphe à l'aide de ces calculs. Mais, il va aussi utiliser d'autres algorithmes (par exemple l'algorithme de Dijkstra) afin d'éviter les noeuds dans les nuages de points en coupant les relations entre de trop lointains voisin. Ainsi, les plus courts chemins entre chaque voisin sont conservés afin d'éviter les "Manifold" (soit l'écrasement des données) pour enfin utiliser un algorithme d'échelle multidimensionnel (MDS) afin de faire un affichage claire, et lisible pour une meilleur visualisation des données.

Preprocessing

La phase de preprocessing consiste à réorganiser notre dataset en sélectionnant les données pertinentes, afin de l'alléger et permettre une classification plus efficace. Il existe pour cela différentes méthodes, dont la conformité [10] dépend du type de données à explorer. Pour notre part, comme il s'agit d'une classification multi-classe, l'apprentissage de nos données se fera dans des espaces de grandes dimensions, ce qui peut nous conduire à ce qu'on appelle un fléau de dimensionnalité [Réf. nécessaire]. On va alors commencer par procéder à une réduction de dimension. En effet, étant donné que notre base de données se résume à des images réparties en plusieurs classes, cela permettra de simplifier la visualisation de la structure du nuage de points qu'on aura obtenu à partir des variables (features) qu'on aura alors sélectionnées car jugées les mieux représentatives de l'information. Pour ce faire, on effectuera tout d'abord une sélection de variables (feature selection) dont la méthode [11] adoptée sera due au choix fait sur le critère de la pertinence du sous-ensemble de données considéré. On va pour cela, éliminer les données qui n'ont pas d'impact voire très peu sur la classification. Ainsi, si on retrouve deux features identiques ou présentant des valeurs à 0 celles-ci ne seront pas prises en compte.

Modelling

La modélisation prédictive est un processus qui utilise l'exploration de données et la probabilité pour prévoir les résultats. Chaque modèle est constitué d'un certain nombre de predicteurs [7], qui sont des variables susceptibles d'influencer les résultats futurs. Une fois que les données ont été recueillies pour les prédicteurs pertinents, un modèle statistique est formulé. Le modèle peut utiliser une équation linéaire simple ou d'un réseau neuronal complexe. À mesure que des données supplémentaires deviennent disponibles, le modèle d'analyse statistique est validé ou révisé.

Pour notre challenge, qui est un problème de classification , on a décidé, après plusieurs recherches et comparaisons entre les méthodes, d'utiliser l'algorithme du k-nearest neighbor (K-NN) qui est une une méthode non paramétrique utilisée pour la classification [8]. L'entrée se compose des k exemples d'apprentissage les plus proches dans l'espace des caractéristiques et la sortie est une appartenance à une classe. Un objet est classé par un vote majoritaire de ses voisins, l'objet étant assigné à la classe la plus commune parmi ses k plus proches voisins (k est un entier positif, typiquement petit). Si k = 1, l'objet est simplement affecté à la classe de ce voisin le plus proche.

À mesure que la taille de l'ensemble de données d'apprentissage approche l'infini, cette méthode garantit un taux d'erreur inférieur au double du taux d'erreur minimal réalisable étant donné la distribution des données.

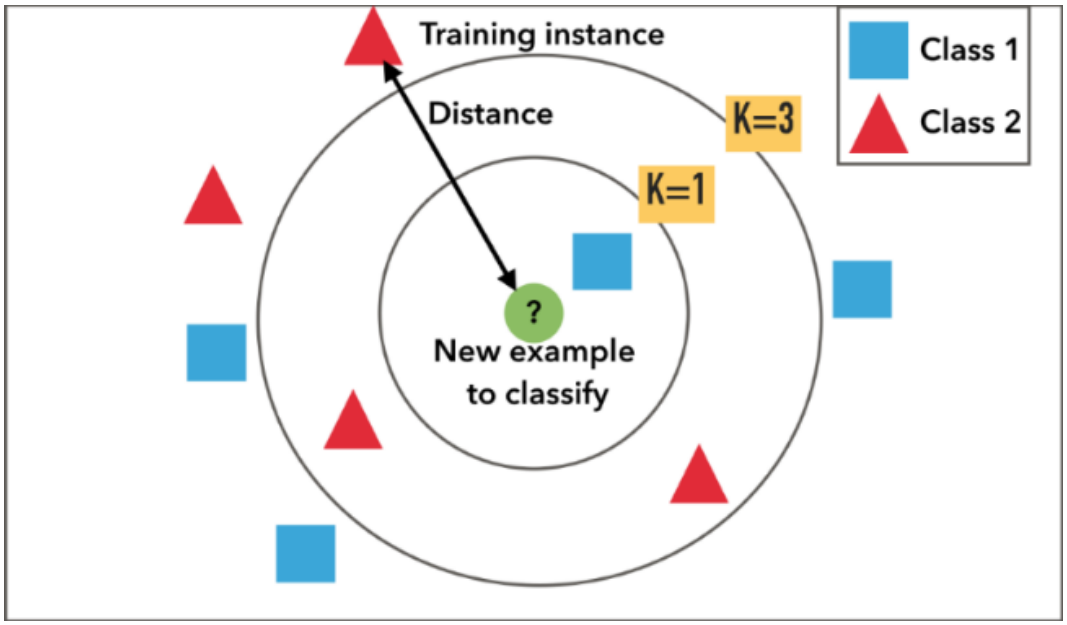
Resultats préliminaires

Annexes

- **Annexe 1**
  [9] : Pseudo code pour le K-NN (Modeling)

```
"""
    @param k : for the nearest neighbor
    @param distances : a fonction that return the distance between the prediction value and k
"""
def find_neighbors(distances, k):
    return distances[0:k]

# votes of neighbors
def find_response(neighbors, classes):
    votes = [0] * len(classes)

    for instance in neighbors:
        for ctr, c in enumerate(classes):
            if instance[-2] == c:
                votes[ctr] += 1

    return max(enumerate(votes), key=itemgetter(1))

# the knn fonction predict the class of the value by averaging the k neighbors
def knn(training_set, test_set, k):
    distances = []
    dist = 0
    limit = len(training_set[0]) - 1

    # generate response classes from training data
    classes = get_classes(training_set)

    try:
        for test_instance in test_set:
            for row in training_set:
                for x, y in zip(row[:limit], test_instance):
                    dist += (x-y) * (x-y)
                distances.append(row + [sqrt(dist)])
                dist = 0

            distances.sort(key=itemgetter(len(distances[0])-1))

            # find k nearest neighbors
            neighbors = find_neighbors(distances, k)

            # get the class with maximum votes
            index, value = find_response(neighbors, classes)

            # Display prediction
            print('The predicted class for sample ' + str(test_instance) + ' is : ' + classes[index])
            print('Number of votes : ' + str(value) + ' out of ' + str(k))

            # empty the distance list
            distances.clear()

    except Exception as e:
      print(e)
```

- **Annexe 2**
  Voici un exemple de K-NN pour mieux comprendre cette méthode:



On a ici deux classes d'éléments (carrés bleus et triangles rouges), et nous essayons de placer un nouvel élément (représenté par un cercle vert),l'élement inconnu, dans l'une de ces deuux classes, on va regarder les voisins les plus proches et voter par exemple pour k = 1, le nouvel exemple est classé comme classe 1, pour k = 3, le nouvel exemple serait placé dans la classe 2 et ainsi de suite jusqu'à trouver la classe à laquelle appartient cet élément.

- **Annexe 3** Methode de desarchivage

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

Bibliographie

[1] : Mariusz Bojarski et al. End-to-End Learning for Self-Driving Cars. arXiv:1604, 2016. arXiv:1604.07316.
[2] : Lex Fridman. End-to-End Learning from Tesla Autopilot Driving Data. GitHub : lexfridman/deeptesla.
[3] : Shai Shalev-Shwartz, Shaked Shammah, Amnon Shashua. Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. arXiv:1610, 2016. arXiv:1610.03295.
[4] : NVIDIA. The AI Car Computer for Autonomous Driving. NVIDIA's website.
[5] : Michael G. Bechtel et al. DeepPicar: A Low-cost Deep Neural Network-based Autonomous Car. arXiv:1712, 2018. arXiv:1712.08644.
[6] : CIFAR10, Alex Krizhevsky, 2009. Learning Multiple Layers of Features from Tiny Images.
[7] : Margaret Rouse. Predictive Modeling. definition/predictive-modeling.
[8] : Altman, N.S.(1992). _"An introduction to kernel and nearest-neighbor nonparametric regression" Tandfonline: _ /doi/abs/10.1080/00031305.1992.10475879.
[9] : div3125. k-nearest-neighbors. GitHub : div3125/k-nearest-neighbors.
[10] : Claude Alain Saby. Study of Dimensionality Reduction Methods for Data Visualization. Résumé
[11] : I. Guyon and A. Elisseeff. An introduction to variable and feature selection. Journal of Machine Learning Research, 2003. Abstract
[12] : Laurens J.P. van der Maaten et Hinton, G.E.. _Visualizing High-Dimensional Data Using t-SNE _. Journal of Machine Learning Research, vol. 9, novembre 2008.