Hadrien Bonato-Pape
Anthony Barna

# Compte rendu
# Partie 1
# TP classification d'images

## Introduction

Tout d'abord il est essentiel de préciser d'où nous partons. Nous avons récupéré le script *Python* fourni par Mme Lingrand.

Nous avons choisi des catégories d'images. Nous différencierons des **motos de course** (moto de grands prix), et des **formule 1**.

Après avoir soigneusement choisi nos images d'étude, nous avons pu passer au cœur du sujet.

## Recherche du meilleur k grâce au score F1 sur les données de validation avec classification par régression linéaire

Afin de trouver la meilleure valeur de k, il est nécessaire de pouvoir évaluer la qualité de l'apprentissage. Nous utilisons donc le score F1 comme élément de comparaison.

Pour obtenir ce score nous ajouterons au code la fonction f1_score de sklearn qui permet de calculer la différence entre le résultat attendu et ce que l'on obtient après l'apprentissage par l'algorithme K-Means.

```
sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary',
sample_weight=None, zero_division='warn')
```

Afin de valider les données nous avons entraîné notre algorithme avec des données d'entraînement (train), nous l'avons entraîné grâce au code fourni par Mme Lingrand puis avons enregistré les objets entraînés pour pouvoir les valider et les tester à posteriori.

La mesure du score F1 se fera grâce à un nouveau script que nous avons écrit : il permet d'essayer de nouvelles données qu'il ne connaît pas et nous verrons comment il les classifie. Cela permet de calculer le score F1 pour en déduire le meilleur k (script en annexe 1).

Nous avons 40 images de chaque classe pour l'entraînement, 10 de chaque pour la validation et 10 pour le test (images envoyées au rendu images).

Afin d'automatiser l'entraînement et la validation nous avons créé des fonctions (avec les chemins en dur). Il n'y a donc plus qu'à renseigner le k souhaité (scripts en annexe 2 et 3).

Les résultats suivants sont obtenus pour les différentes valeurs de k :

| K (nb of clusters) | 5 | 7 | 10 | 11 |
|---|---|---|---|---|
| TRAIN F1 score | 0,69 | 0,66 | 0,72 | 0,52 |

| 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|
| 0,55 | 0,47 | 0,47 | 0,47 | 0,57 | 0,42 |

| 18 | 19 | 20 | 22 | 25 | 30 |
|---|---|---|---|---|---|
| 0,44 | 0,6 | 0,4 | 0,26 | 0,0 | 0,63 |

| 40 | 50 | 100 | 200 |
|---|---|---|---|
| 0,66 | 0,63 | 0,625 | 0,5 |

Nous remarquons que les résultats sont assez faibles, et très variables, cela doit être à cause des images choisies. Les meilleurs résultats sont obtenus pour k =10. Mais à la vue des résultats plutôt faibles nous avons cherché à obtenir de meilleurs résultats en changeant les images d'entraînement.

Nous avons donc décidé de trier à nouveau les images et d'en choisir de nouvelles : plus proches les unes des autres avec le même angle de prise de vue si possible pour les deux catégories d'objets photographiés. Ces nouvelles images sont disponibles sur mon respository GitHub.

Cette fois ci nous avons 20 images de chaque classe pour l'entraînement, 10 de chaque pour la validation et 10 pour le test.

Nous avons choisi pour ce nouvel apprentissage uniquement des images d'engins dans le même sens avec le moins de fond possible, quitte à retourner horizontalement des images pour avoir le même sens des véhicules. Étant donné la grande quantité d'éléments visuels que ces images renvoient, entre les couleurs et les stickers, les multiples pièces… Cela fait beaucoup d'éléments pouvant perturber la description de l'image via les SIFT, et donc l'apprentissage.

Après avoir uniformisé les nouvelles images nous entraînons de nouveau notre algorithme.
Une fois entraîné nous pouvons calculer notre score F1 pour chaque K via nos images de validation.

Les résultats obtenus sont assez mauvais :

| K (nb of clusters) | 5 | 7 | 10 | 11 |
|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| TRAIN F1 score | | 0,39 | 0,42 | 0,66 | 0,57 |

| | | | | | |
|---|---|---|---|---|---|
| 12 | 13 | 14 | 15 | 16 | 17 |
| 0,57 | 0,66 | 0,53 | 0,625 | 0,46 | 0,714 |

| | | | | | |
|---|---|---|---|---|---|
| 18 | 19 | 20 | 22 | 25 | 30 |
| 0,53 | 0,53 | 0,57 | 0,57 | 0,5 | 0,57 |

| | | | |
|---|---|---|---|
| 40 | 50 | 100 | 200 |
| 0,42 | 0,36 | 0,36 | 0,46 |

Le problème vient soit du fait qu'on ait réduit le nombre d'images d'entraînement soit à cause de la complexité des images qui reste très importante.

Donc par curiosité nous avons créé un dernier entraînement avec toutes les images (images du premier entraînement et du second). Nous avons testé avec les images de validation de la série « triée ».

Cela représente (en supprimant les images en double) :

- 130 images de train (65 par catégorie),
- 20 images de test (10 par catégorie),
- 20 images de validation (10 par catégorie).

Il est évident que l'apprentissage a été long (plus d'une heure et demie).

Malgré la grande quantité d'images les tests ne sont pas concluants avec un score F1 ne dépassant pas les **0,46** pour un k de **25**. Le score F1 tombe même à 0 pour un k de 200 ou 14.

Notre hypothèse quant à des résultats si faibles est que la complexité visuelle des images est trop importante pour ce genre d'algorithmes. C'est-à-dire que le nombre d'éléments visuels qui composent ces images est trop important entres toutes les pièces qui composent ces engins, les couleurs des carrosseries et des carénages ou encore les stickers de sponsors . Cela pose un problème pour la description qui est brouillée ou trop complexe.

Une autre méthode de classification est nécessaire. Ou bien une autre méthode de description ?

Nous allons essayer avec la méthode de classification SVM, les séparateurs à vaste marge afin de pouvoir comparer ces deux méthodes.

## Recherche de la meilleure valeur de C (avec K = 50) via le score F1

Dans un premier temps nous avons écrit un nouveau script permettant l'apprentissage du nouvel algorithme de classification, le SVM.

Nous avons donc écrit un script pour simplifier la tache et juste attendre le calcul des scores après l'entraînement automatisé lui aussi.

C'est-à-dire que nous avons une fonction dont le rôle est l'apprentissage, qui prend en paramètre la valeur du k et de C. Si on a déjà calculé le KMEAN correspondant on le reprend, sinon on le calcule. Puis on entraîne le classificateur SVM en fonction de la valeur de C choisie. On a aussi au fil du TP essayé tous les Kernels pour voir ce qu'ils apportaient en fonction des cas.

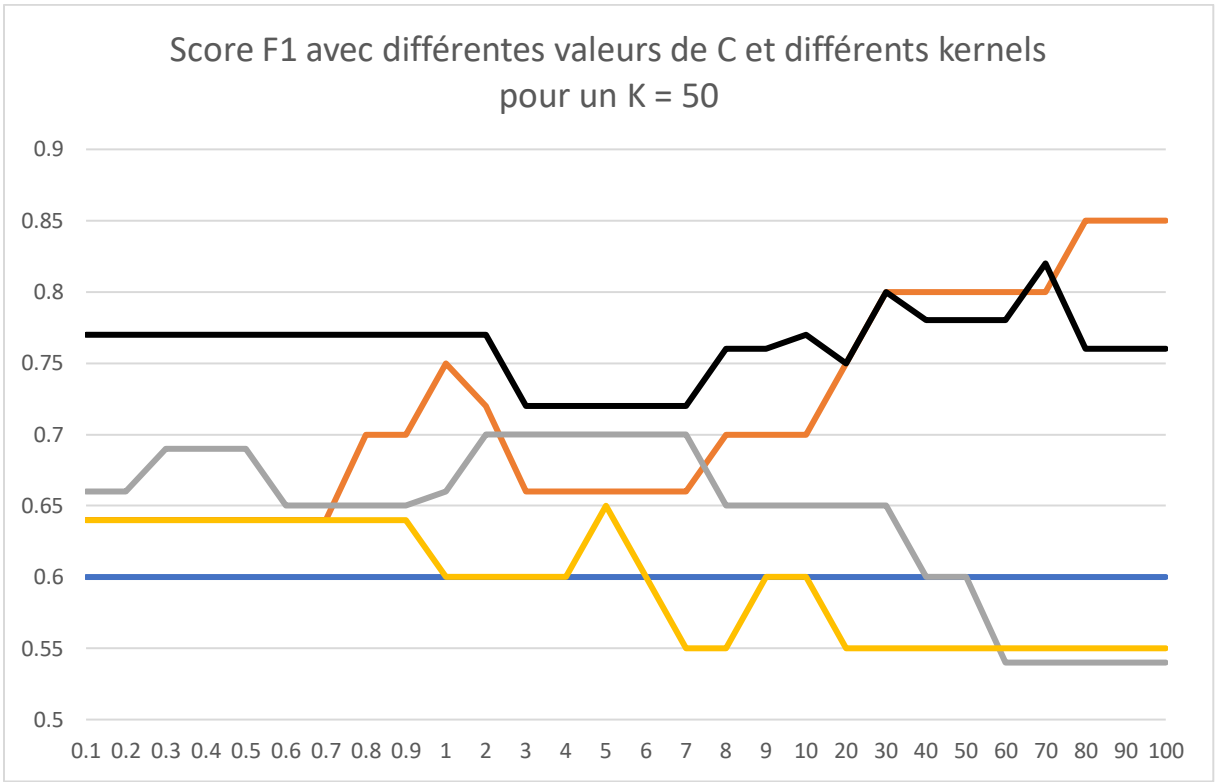
Puis nous avons une deuxième fonction qui va elle aussi prendre en paramètre K et C pour retrouver les fichiers d'objets entraînés préalablement. Nous allons ensuite pouvoir vérifier notre score F1 en soumettant de nouvelles images (nos images de validation, les mêmes que dans la première partie).

Ensuite nous avons fait une boucle pour essayer toutes les valeurs de C (dans un intervalle donné et avec un pas à indiquer aussi) en fonction du K indiqué.

Vous trouverez le code complet dans l'annexe 4.

Pour nos images « d'origines » (celles envoyées sur Moodle) avec un k = 50 comme demandé et le kernel « *linear* » on a essayé les valeurs de C suivantes : 0 à 1 avec un pas de 0,1 puis de 1 à 10 avec un pas de 10, puis de 10 à 100 avec un pas de 10.

Nous avons relevé les valeurs suivantes :




Orange = kernel 'rbf'

Bleu = kernel ‘linear’
Gris = kernel ‘poly’
Jaune = kernel ‘sigmoid’
Noir en pointillés = Score F1 avec les mêmes valeurs de C pour un K = 10 avec le kernel ‘rbf’ pour SVM.

Nous avons en orange la « meilleure » courbe et donc le meilleur apprentissage (avec le kernel ‘*rbf*’).
Le meilleur score F1 est obtenu à partir de C = 80, pour un score F1 de 0,85 il stagne jusqu’à C = 400, puis rechute. Nous avons donc des marges « dures », en dessous d’un C de 0.1 les scores de F1 sont de 0.0, ou presque la plupart du temps. Nous avons donc choisi de les exclure par gain de temps. Les marges « moles » donnent trop de libertés et donc d’erreurs de classification alors que nos données sont déjà très proches . Les données deviennent alors impossibles à classifier.

Nous avons voulu essayer avec le meilleur k possible, c’est-à-dire K = 10 (celui avec lequel nous avons eu les meilleurs résultats pour la classification par régression logistique).

Et les scores ne sont pas meilleurs même s’ils sont plus stables. Avec le meilleur score F1 a : 0,82 (courbe noire en pointillés).

Nous avons aussi souhaité essayer les différentes valeurs de C sur les précédents fichiers d’images (les images « triées » et toutes les images trouvées).

Sur les images triées les résultats sont toujours très faibles pas un seul score F1 a plus de 0,5. Pour les images « non triées » avec toutes les images qu’on a pu rassembler on avait le meilleur score F1 (0,46) à K = 25, ici avec le kernel ’rbf’ et plusieurs essais la meilleure valeur de C trouvée est 0,83 pour C = 2. Le séparateur à vastes marges (SVM) a quasiment doublé notre capacité de classification avec les mêmes images d’entraînement.

## Conclusion et comparaison des 2 méthodes de classification

Les deux méthodes de classification ont des performances différentes. Sur nos images la classification SVM a sensiblement amélioré nos résultats de score F1. Faisant passer notre meilleur score F1 de 0,72 à 0,85.

La totalité des codes ainsi que les fichiers d’objets déjà entraînés sont disponibles sur mon GitHub à cette adresse : https://github.com/Hadrien-Bonato-Pape/SSII/tree/master/TP-FINAL-SSII-RECO-IMAGE

Nous avons pris à cœur ce TP, ainsi que cette matière et portons un grand intérêt à cette dernière, mais aussi à la poursuite du cursus autour du machine learning.

## Annexes :

## 1 - Script de classification d'images inconnues via la régression logistique (calcul du score F1) :

```
import glob
from sys import argv
import cv2
import pickle

import shutil
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import f1_score

k1 = int(argv[1])

cat1 = argv[2]
cat2 = argv[3]

baryName = argv[4]

listImg=glob.glob(cat1+"/*.jpeg")
tmpa = len(listImg)
listImg += glob.glob(cat2+"/*.jpeg")

print("Catégorie " + cat1 + "est normalement 0.")
print("Catégorie " + cat2 + "est normalement 1.")

lesSift = np.empty(shape=(0, 128), dtype=float) # array of all SIFTS from all images
dimImg = [] # nb of sift per file
groundTruth = [0]*tmpa # result we would like to reach
tmpb = len(listImg)-tmpa
groundTruth += [1]*tmpb

for s in listImg:
```

```
    print("###",s,"###")
    image = cv2.imread(s)
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp,des = sift.detectAndCompute(gray,None)
    print("SIFT: ", len(kp))
    dimImg.append(len(des))
    lesSift = np.append(lesSift,des,axis=0)

with open(baryName+'k.bary',  'rb') as input:
    km1 = pickle.load(input)

bows = np.empty(shape=(0,k1),dtype=float)
km1.predict(lesSift)

i = 0
for nb in dimImg:
    tmpBow = [0]*k1
    j = 0
    while j < nb:
        tmpBow[km1.labels_[i]] += 1
        j+=1
        i+=1
    copyBow = tmpBow.copy()
    bows = np.append(bows, [copyBow], 0)

with open(baryName+'L.logr',  'rb') as input:
    logisticRegr = pickle.load(input)



res = logisticRegr.predict(bows)

print(res)
print(groundTruth)

score = logisticRegr.score(bows, groundTruth)
```

```
print("f1 score = ",f1_score(groundTruth, res, average='binary'))
print("train score = ", score)
```

## 2 - Script d'entrainement via la régression logistique « automatisé » :

```
import glob
from sys import argv
import cv2
import pickle

import shutil
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import f1_score

cat1 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/images/train/formule1"
cat2 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/train/motoGrandPrix"

def automatedTrainData(k1):
    baryName = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/bary-logr/k" +
str(k1)

    listImg=glob.glob(cat1+"/*.jpeg")
    tmpa = len(listImg)
    listImg += glob.glob(cat2+"/*.jpeg")

    lesSift = np.empty(shape=(0, 128), dtype=float) # array of all SIFTS from all images
    dimImg = [] # nb of sift per file
    groundTruth = [0]*tmpa # result we would like to reach
    tmpb = len(listImg)-tmpa
    groundTruth += [1]*tmpb

    for s in listImg:
```

```
    image = cv2.imread(s)
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp,des = sift.detectAndCompute(gray,None)
    dimImg.append(len(des))
    lesSift = np.append(lesSift,des,axis=0)

#BOW initialization
bows = np.empty(shape=(0,k1),dtype=float)

# everything ready for the k-means
kmeans1 = KMeans(n_clusters=k1, random_state=0).fit(lesSift)

with open(baryName+'k.bary', 'wb') as output:
    pickle.dump(kmeans1, output, pickle.HIGHEST_PROTOCOL)

bary1 = kmeans1.cluster_centers_;

#writing the BOWs for second k-means
i = 0
for nb in dimImg: # for each sound (file)
    tmpBow = [0]*k1
    j = 0
    while j < nb: # for each SIFT of this sound (file)
        tmpBow[kmeans1.labels_[i]] += 1
        j+=1
        i+=1
    copyBow = tmpBow.copy()
    bows = np.append(bows, [copyBow], 0)
#if verbose:
    # print("BOWs : ", bows)

#ready for the logistic regression
logisticRegr = LogisticRegression(max_iter=1000)
logisticRegr.fit(bows, groundTruth)
with open(baryName+'L.logr', 'wb') as output:
```

```
        pickle.dump(logisticRegr, output, pickle.HIGHEST_PROTOCOL)

    res = logisticRegr.predict(bows)
    score = logisticRegr.score(bows, groundTruth)
    print("k = ", k1)
    print("train score = ", score)

automatedTrainData(5)
automatedTrainData(7)
automatedTrainData(10)
automatedTrainData(11)
automatedTrainData(12)
automatedTrainData(13)
automatedTrainData(14)
automatedTrainData(15)
automatedTrainData(16)
automatedTrainData(17)
automatedTrainData(18)
automatedTrainData(19)
automatedTrainData(20)
automatedTrainData(22)
automatedTrainData(25)
automatedTrainData(30)
automatedTrainData(40)
automatedTrainData(50)
automatedTrainData(100)
automatedTrainData(200)
```

## 3 - Script de classification (classification grâce à la régression logistique) d'images inconnues « automatisé » (calcul du score F1 grâce aux images de validation) :

```
import glob
from sys import argv
import cv2
import pickle
```

```
import shutil
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import f1_score

cat1 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/validation/formule1"
cat2 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/validation/motoGrandPrix"

def automatedValidateData(k1):
    baryName = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/bary-logr/k" +
str(k1)

    listImg=glob.glob(cat1+"/*.jpeg")
    tmpa = len(listImg)
    listImg += glob.glob(cat2+"/*.jpeg")

    lesSift = np.empty(shape=(0, 128), dtype=float) # array of all SIFTS from all images
    dimImg = [] # nb of sift per file
    groundTruth = [0]*tmpa # result we would like to reach
    tmpb = len(listImg)-tmpa
    groundTruth += [1]*tmpb

    for s in listImg:
        image = cv2.imread(s)
        gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        sift = cv2.xfeatures2d.SIFT_create()
        kp,des = sift.detectAndCompute(gray,None)
        dimImg.append(len(des))
        lesSift = np.append(lesSift,des,axis=0)

    with open(baryName+'k.bary',  'rb') as input:
        km1 = pickle.load(input)
```

```
    bows = np.empty(shape=(0,k1),dtype=float)
    km1.predict(lesSift)

    i = 0
    for nb in dimImg:
        tmpBow = [0]*k1
        j = 0
        while j < nb:
            tmpBow[km1.labels_[i]] += 1
            j+=1
            i+=1
        copyBow = tmpBow.copy()
        bows = np.append(bows, [copyBow], 0)

    with open(baryName+'L.logr',  'rb') as input:
        logisticRegr = pickle.load(input)


    res = logisticRegr.predict(bows)

    print("k = " + str(k1))
    print(res)
    print(groundTruth)

    score = logisticRegr.score(bows, groundTruth)
    print("f1 score = ",f1_score(groundTruth, res, average='binary'))
    print("train score = ", score)


automatedValidateData(5)
automatedValidateData(7)
automatedValidateData(10)
automatedValidateData(11)
automatedValidateData(12)
automatedValidateData(13)
```

```
automatedValidateData(14)
automatedValidateData(15)
automatedValidateData(16)
automatedValidateData(17)
automatedValidateData(18)
automatedValidateData(19)
automatedValidateData(20)
automatedValidateData(22)
automatedValidateData(25)
automatedValidateData(30)
automatedValidateData(40)
automatedValidateData(50)
automatedValidateData(100)
automatedValidateData(200)
```

## 4 - Script de d'apprentissage et de classification (classification grâce au SVM) d'images inconnues « automatisé » (calcul du score F1 grâce aux images de validation) :

```
from sklearn import svm
import glob
from sys import argv
import cv2
import pickle
import shutil
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import f1_score

def automatedTrainDataForSVM(c1, k1, toRecomputeKMEANS, verbose, minimal):
    if(minimal):
        print("c = ", c1)
        print("k = ", k1)
```

```
    baryName = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/bary-logr/k" +
str(k1)

    listImg=glob.glob(cat1+"/*.jpeg")
    tmpa = len(listImg)
    listImg += glob.glob(cat2+"/*.jpeg")

    lesSift = np.empty(shape=(0, 128), dtype=float) # array of all SIFTS from all images
    dimImg = [] # nb of sift per file
    groundTruth = [0]*tmpa # result we would like to reach
    tmpb = len(listImg)-tmpa
    groundTruth += [1]*tmpb

    for s in listImg:
        image = cv2.imread(s)
        gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        sift = cv2.xfeatures2d.SIFT_create()
        kp,des = sift.detectAndCompute(gray,None)
        dimImg.append(len(des))
        lesSift = np.append(lesSift,des,axis=0)

    if(toRecomputeKMEANS):
        if(verbose):
            print("KMEAN FIT")
        km1 = KMeans(n_clusters=k1, random_state=0).fit(lesSift)
        with open(baryName+'k.bary', 'wb') as output:
            pickle.dump(km1, output, pickle.HIGHEST_PROTOCOL)
    else:
        with open(baryName+'k.bary',  'rb') as input:
            km1 = pickle.load(input)

    bows = np.empty(shape=(0,k1),dtype=float)
    km1.predict(lesSift)

    i = 0
    for nb in dimImg:
```

```
        tmpBow = [0]*k1
        j = 0
        while j < nb:
            tmpBow[km1.labels_[i]] += 1
            j+=1
            i+=1
        copyBow = tmpBow.copy()
        bows = np.append(bows, [copyBow], 0)

    #ready for the SVM

    if(verbose):
        print("SVM FIT")

    classif = svm.SVC(c1,kernel='rbf')
    classif.fit(bows, groundTruth)

    with open(baryName + str(c1) + 'SVM.svm', 'wb') as output:
        pickle.dump(classif, output, pickle.HIGHEST_PROTOCOL)

    if(verbose):
        predict = classif.predict(bows)
        vectors = classif.support_vectors_
        score = classif.score(bows, groundTruth)
        print('Prediction class for bows', predict)
        print('Support vectors: ', vectors)
        print("Train score = ", score)

    if(minimal):
        print(" ")


def automatedValidateDataSVM(c1, k1):
    baryName = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/bary-logr/k" +
str(k1)
```

```
listImg=glob.glob(cat1+"/*.jpeg")
tmpa = len(listImg)
listImg += glob.glob(cat2+"/*.jpeg")

lesSift = np.empty(shape=(0, 128), dtype=float) # array of all SIFTS from all images
dimImg = [] # nb of sift per file
groundTruth = [0]*tmpa # result we would like to reach
tmpb = len(listImg)-tmpa
groundTruth += [1]*tmpb

for s in listImg:
    image = cv2.imread(s)
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp,des = sift.detectAndCompute(gray,None)
    dimImg.append(len(des))
    lesSift = np.append(lesSift,des,axis=0)

with open(baryName+'k.bary',  'rb') as input:
    km1 = pickle.load(input)

bows = np.empty(shape=(0,k1),dtype=float)
km1.predict(lesSift)

i = 0
for nb in dimImg:
    tmpBow = [0]*k1
    j = 0
    while j < nb:
        tmpBow[km1.labels_[i]] += 1
        j+=1
        i+=1
    copyBow = tmpBow.copy()
    bows = np.append(bows, [copyBow], 0)

with open(baryName  + str(c1) +  'SVM.svm',  'rb') as input:
```

```
        classif = pickle.load(input)


    res = classif.predict(bows)
    print(res)
    print(groundTruth)
    f1Score = f1_score(groundTruth, res)
    print("F1 score = ", f1Score)
    print(" ")
    return f1Score

#-------------------------------------------------------------------------------------------------------------
Pseudo Main
toRecomputeKMEANS = False
verbose = False
minimal = True
c1 = 1
k1 = 50
f1Scores = []
interval = 1
maxC1 = 10

while(c1 < maxC1):
    print("-----------------------------------------")
    print("Training SVM")
    cat1 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-IMAGE/images/train/formule1"
    cat2 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/train/motoGrandPrix"
    automatedTrainDataForSVM(c1, k1, toRecomputeKMEANS, verbose, minimal) #Training
    cat1 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/validation/formule1"
    cat2 = "/Users/hbp/Documents/GitHub/SSII/TP-FINAL-SSII-RECO-
IMAGE/images/validation/motoGrandPrix"
    print("Validate with F1 score (with validate data -> pictures)")
    f1Scores.append(automatedValidateDataSVM(c1, k1))
    c1 = c1 + interval
```

```
maxF1 = max(f1Scores)
i = 0
intervalRecherche = interval
print("----------------------------------------")
print("Find best F1 SCORE : ")
print("Best F1 SCORE = ", maxF1)
print("For values of C : ")

while(intervalRecherche < maxC1):
    if(f1Scores[i] == maxF1):
        print(intervalRecherche)
    i = i + 1
    intervalRecherche = intervalRecherche + interval

print("All values of F1")
for s in f1Scores:
    print(s)
```