Telecom ParisTech

Apprentissage pour l'image et la reconnaissance d'objets

IMA 205

Rapport Challenge

Author :
de Sousa Silva, Raul Alfredo

Professor :
Gori, Pietro

Livré le : 21 avril 2019

Table des matières

1 Objectifs

Ce rapport a pour objectif de montrer les méthodes d'apprentissage qui ont été évalués pour l'exécution du challenge d'IMA 205, où l'objectif était de créer un algorithme capable de classer les images de lésions cutanées correctement entre cancérigènes et non-cancérigènes.

2 Introduction

Les méthodes pour apprentissage ont gagné beaucoup de place dans les derniers années surtout dans les domaines d'aide à un spécialiste, comme par exemple, le domaine médical.

Pour mettre en pratique certains de ces algorithmes, nous avons un challenge à exécuter à la fin de ce cours : Un ensemble de données avec 1000 images nous est mise à disposition, d'où 700 ont été déjà classées et 300 sont à classer à partir de l'apprentissage faite avec les premières.

On parle des images de lésions cutanées, entre elles nous avons des exemples d'images cancérigènes et non-cancérigènes. L'objectif c'est d'obtenir la meilleure classification possible de ces 300 dernières images en utilisant n'importe quelle des méthodes appris pendant le cours. La qualité de la classification est, alors, mesurée par la corrélation de Matthews, donné par :

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{1}$$

Où,

- TP : proportion d'échantillons positives bien classés (comme positives) ;
- TN : proportion d'échantillons négatives bien classés (comme négatives) ;
- FP : proportion d'échantillons négatives mal classés (comme positives) ;
- FN : proportion d'échantillons positives mal classés (comme négatives) ;

Alors, pour essayer de faire le meilleur score possible seront testés 4 méthodes (les justificatives de pourquoi les essayer et que-est-ce que nous devrons attendre sera explicité à chaque session) :

— K-plus proches voisins (KPPV) ;
— Support Vector Machines (SVM) ;
— Arbres de décision ;
— Multi-Layer Perceptron (MLP) ;

L'idée c'est d'augmenter la complexité de l'algorithme d'apprentissage peu à peu dans l'espoir d'avoir à chaque fois un algorithme plus performant et robuste, au fil des expérimentations. A la fin, nous allons discuter de la performance des différentes méthodes utilisées.

3 L'extraction des attributs

Pour que nous puissions faire la classification des images, il faut avoir des attributs que nous permettrons d'appliquer l'algorithme de classification. Pour l'instant, nous n'avons que des images des lésions et les cartes de segmentations. Alors, nous devrons chercher certaines

caractéristiques qui puissent être intéressantes à détecter une lésion cancérigène (ou non-cancérigène).

Basée sur le travail de [1], nous avons obtenu 433 attributs liées à la forme, à la couleur et à la texture des images. En particulier, nous avons pris 13 attributs de forme, 348 attributs de couleur et 72 attributs de texture des 437 proposés par l'article original (certains attributs de forme ont été ajoutés à ceux qui ont été proposés par l'article dans l'espoir de créer des attributs non-linéaires qui puissent améliorer la classification). Alors, une liste un peu plus détaillée de ces attributs est présenté ci-dessous, la définition précise de chacune peut être facilement trouvé sur l'article original.

Attributs de forme

- Surface
- Périmètre
- Excentricité
- Asymétrie 1
- Longueur de l'axe majeur
- 3 moments d'ordre 2 ((0,2),(2,0),(1,1))
- Diamètre équivalente
- Ratio d'aspect
- Orientation de l'axe majeur
- Asymétrie 2
- Longueur de l'axe mineur

Attributs de couleur

Pour le calcul des attributs de couleur, l'image est projetée dans six espaces de couleurs : RGB, rgb (RGB normalisé), HSV, I1/2/3 (Ohta space), l1/2/3 and CIE L*u*v*. Dans chaque canal de chaque espace couleur ont été prises les attributs décrites ci-dessous. Alors, à l'aide de la carte de segmentation, les images sont découpées en trois parties d'intérêt : la lésion elle-même, la périphérie interne à la lésion et la périphérie externe à la lésion. Alors, nous avons 6 (espaces couleurs) $\times$ 3 (canaux) $\times$ 3 (régions d'intérêt) = 54 régions d'où extraire des attributs.

- Moyenne et écart-type (D'ailleurs, les rapports et les différences des 2 statistiques sur les 3 régions ont également été calculés : (externe / interne), (externe / lésion), (interne / lésion), (externe - interne), (lésion - externe), et (lésion - interne)). (324)
- Asymétrie de couleur (Exclusivement en RGB) (6)
- Distance des centroïdes (1 pour chaque canal de chaque espace couleur) (18)

Attributs de texture

Afin de quantifier la texture présente dans une lésion, un ensemble de descripteurs statistiques de texture basées sur la co-occurrence de la matrice de niveau de gris (GLCM) ont été utilisés. Bien que de nombreuses statistiques puissent être dérivées du GLCM, huit statistiques invariantes par décalage de niveau de gris ont été utilisés. D'ailleurs, les rapports et les différences des statistiques sur les 3 régions ont également été calculés : (externe / interne), (externe / lésion), (interne / lésion), (externe - interne), (lésion - externe), et (lésion - interne).

- Probabilité maximale
- Énergie
- Entropie
- Dissimiabilité
- Contraste
- Différence inverse
- Moment de différence inverse
- Corrélation.

4 Mise en ouvre

4.1 Pré-traitement

Pour permettre une classification adéquate sans être biaisé par l'extension où l'ordre de grandeur de chaque attribut, il faut avant tout, faire une normalisation des attributs. Cela est très facilement exécuté par les fonctions mises en place par la bibliothèque *sklearn*. Avec la fonction *StandardScaler* de *sklearn.preprocessing* la normalisation est faite en trois lignes de code.

Aussi pour accélérer l'entraînement et pour ne pas garder des attributs qui ne sont pas importants pour décrire les images, nous appliquons aussi une réduction de dimensionalité, en particulier nous appliquons une Analyse en Composantes Principales (ACP) pour pouvoir obtenir les caractéristiques qui expliquent à 95% les échantillons de notre ensemble. Cette analyse est aussi facilement faite à partir de la fonction *PCA* de *sklearn.decomposition*.

4.2 KPPV

Pour la mise en oeuvre d'une classification avec KPPV nous n'avons pas grand-chose à faire. D'ailleurs, c'est une bonne référence pour évaluer la qualité des autres méthodes. La classification pour k plus proches voisins c'est aussi assez raisonnable, étant vu que nous attendions certainement que des images de lésion cancérigène (et non-cancérigènes) aient des caractéristiques en commun.

Alors, pour appliquer une classification pour K-plus proche voisins nous devrons utiliser la bibliothèque *sklearn* qui contient entre autres méthodes de classification, le KPPV. Il nous reste donc d'appliquer la fonction *KNeighborsClassifier* aux échantillons avec le nombre de voisins désirés.

Pour améliorer la classification, nous pouvons faire une évaluation du score en fonction du hyperparamètre k (nombre de voisins) et ainsi, optimiser la classification. Cette amélioration est faite avec les fonctions *cross_val_score*, *GridSearchCV* and *KFold* de *sklearn.model_selection*.

À la fin, nous pouvons gagner un peu plus de robustesse à l'aide d'une technique de boosting, qui permet d'améliorer sensiblement la qualité de la classification par l'utilisation de plusieurs classificateurs faibles, concept vu en IMA206, qui vont, chacun contribuer pour la classification avec un vote. Pour des classificateurs simples comme les KPPV et arbres de décision, qui ne demandent pas un grand effort de calcul, une telle stratégie pourrait être assez efficace.

4.3 SVM

La méthode du SVM est beaucoup moins évidente et peut être un peu naïve si certains changements ne sont pas faits. En particulier, la classification par SVM suppose la possibilité de diviser l'espace avec une droite, ce qui n'est pas tout à fait vrai. Mais quand même nous pouvons trouver une droite qui minimise l'erreur de la classification de telle sorte qu'on puisse avoir une classification acceptable.

Pour appliquer le SVM nous utilisons aussi la bibliothèque *sklearn* avec la fonction *LinearSVC*. Le hyperparamètre C de régularisation du SVM (pour qu'il puisse admettre une solution où la division de l'espace ne soit pas parfaite) est réglé avec une validation croisée de la classification avec différentes valeurs du hyperparamètre ainsi comme en KPPV.

4.4 Arbres de décision

Les arbres de décision sont, grosso modo, un raffinement des KPPV de manière un peu plus élégante. Tandis que KPPV va créer des régions où un éventuel nouveau échantillon serait classé comme positif (ou négatif) à cause de la proximité plutôt des exemples positifs (ou négatifs), les arbres de décision vont diviser l'espace en plusieurs régions à partir de seuils sur l'un des attributs de tel manière à minimiser l'erreur de classification.

Alors, les mêmes possibilités d'améliorations du KPPV s'ouvent aux arbres de décision. Les arbres pourraient être réglés par certains hyperparamètres comme le nombre minimum d'échantillons dans chaque "brique" de l'espace ou le nombre maximum de "feuilles", c'est-à-dire divisions subséquentes d'un sous espace. Aussi, le boosting pourrait améliorer la qualité de la classification par le comptage de votes de classification de plusieurs arbres entraînés différemment.

4.5 MLP

Le Perceptron Multi-couches, est une technique de Deep Learning qui a aussi une bonne chance de faire une bonne classification sur les attributs obtenus. Le principe du MLP est l'application de plusieurs couches de perceptrons, chaque couche de taille variée ou tous les perceptrons de la couche antécédente participent de la fonction d'activation de chaque perceptron de la couche actuelle.

L'objectif est d'entraîner les poids des perceptrons à chaque couche pour améliorer la classification de l'ensemble d'entraînemment pour permettre d'améliorer la classification dans l'ensemble de test.

Alors, le MLP n'a pas forcément de hyperparamètres qui peuvent être choisis pour optimiser la classification, mais par contre, l'architecture du réseau est complètement libre, c'est-à-dire que nous pouvons faire n'importe quel nombre de couches et n'importe quelle taille à chaque couche, et même il y a plusieurs possibilités de non-linéarités qui doivent être bien choisies pour garantir de succès de l'algorithme.

Tout ce que nous savons, c'est que le plus grand est le nombre de couches et le plus grand est le nombre de perceptrons à chaque couche, le plus de non-linéarités nous avons, cela complexifie le réseau de neurones, mais augmente aussi la précision de la classification.

Pour mettre en place le MLP nous avons utilisé une architecture en basée sur la bibliothèque *tensorflow* avec deux couches cachés de 256 neurones chacune.

5 Résultats

5.1 KPPV

Avec la mise en place de la classification par KPPV nous faisons un entraînement avec les 700 échantillons du training set, avec tous les nombres impaires de voisins entre 1 et 15 pour pouvoir choisir le meilleur nombre de voisins nécessaires pour classer un échantillon.

La valeur de K obtenu a été de 9 voisins, avec un score *balanced_ accuracy* de 0.64.

En appliquant au test set, le résultat obtenu dans la plateforme kaggle a été d'un score en Corrélation de Matthews de autour 0.14 ce qui signifie (considérant une certaine homogénéité entre les classes) une accuracy de autour 0.57.

En faisant une implémentation de adaboost pour améliorer le score du KPPV avec 21 classifieurs, nous obtenons un *balanced_ accuracy* de 0.99 tandis que sur kaggle la Corrélation de Matthews a été de 0.2.

5.2 SVM

Avec la mise en place de la classification par SVM nous faisons un entraînement avec les 700 échantillons du training set, avec un facteur de régularisation de 'C' : [0.001, 0.01, 0.1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

La valeur de C obtenu a été de 1, avec un score *balanced_ accuracy* de 0.67.

En appliquant au test set, le résultat obtenu dans la plateforme kaggle a été d'un score en Corrélation de Matthews de autour 0.4 ce qui signifie (considérant une certaine homogénéité entre les classes) une *balanced_ accuracy* de autour 0.8.

Pour améliorer la classification, nous mettons en place aussi un SVM non-linéaire à l'aide de la fonction *SVC*. Avec elle et un autre paramètre de courbure 'gamma' ajouté à l'analyse nous arrivons à une *balanced_ accuracy* de 0.67 dans le training et une Corrélation de Matthews de autour 0.40, ce qui signifie une très légère amélioration par rapport au SVM linéaire. Meilleurs paramètres : {'C' : 7, 'gamma' : 0.001}.

5.3 Arbres de décision

Avec le mise en place de la classification par arbres de décisions, sans influence d'aucun hyperparamètre nous obtenons une *balanced_ accuracy* de 1 dans le training, mais une Corrélation de Matthews de autour 0.17 dans le test set selon la plateforme kaggle.

5.4 MLP

En utilisant l'architecture décrite dans la section antérieure, nous avons une apprentissage presque complète dans le training, (accuracy 0.96) mais un score en Corrélation de Matthews de 0.37.

6 Conclusion

L'objectif de ce challenge, outre que familiariser les élèves avec des défis d'un travail en science de données, était de détecter les lésions cancérigènes parmi un ensemble de 1000 images de lésions de peau par l'utilisation de quelques algorithmes d'apprentissage parmi ceux appris dans le cours.

L'extraction d'attributs s'est montré comme un facteur crucial pour permettre une détection efficace des lésions cancérigènes, ce n'est pas toujours fondamental extraire beaucoup d'attributs, mais c'est fondamental extraire de bons attributs. Alors, nous devrons admettre que les attributs extraits n'ont pas forcément été super efficaces et peut-être que le calcul des attributs pourrait être beaucoup plus rapide et souffle si une autre référence était utilisée.

Quant aux algorithmes d'apprentissage, ils ont eu une performance assez moyenne, peut-être limité par les attributs disponibles. Le KPPV nous a donné une bonne idée de base, mais il a montré rapidement qu'il fallait penser à quelque chose de plus complexe pour permettre une prédiction plus qualifiée. Le résultat du boosting effectué a amélioré beaucoup le taux d'apprentissage, mais il a peu developpé en termes de prédiction.

L'utilisation des arbres de décision ont amélioré un peu la qualité de prédiction par rapport aux KPPV, mais il n'a même pas dépassé le KPPV "boosté". Alors, même s'il à eu l'apprentissage la plus efficace, il a créé un overfitting sur l'espace des attributs, ce qui a fini pour limiter la performance de cet algorithme d'apprentissage.

Le SVM a eu alors un très bon résultat de classification, par rapport aux deux premiers. Pourtant, les tentatives d'amélioration de la classification avec le paramètre de courbure ont peu contribué pour l'optimisation du classifieur. Alors, c'est lui que nous gardons comme le meilleur classifieur pour le contexte du challenge.

Le réseau de neurone, le MLP a été peut-être un peu frustrante. Malgré son efficacité proche de celle du MLP (mais même pas supérieur), nous attendions quelque chose de plus robuste et efficace, étant vu qu'il a une tendance à contourner les non-linéarités éventuelles du système. Il est possible que l'architecture a été peu développée et que l'imposition d'autres couches pourrait beaucoup améliorer ce classifieur.

Pour conclure, nous avons eu une bonne occasion de mettre en oeuvre tout ce que nous avons appris dans de cours dans un contexte pratique. Nous sommes confrontés avec les défis et difficultés que nous aurons très probablement dans l'univers professionnel, alors, cela nous a permis de les avoir déjà en tête et concevoir certaines idées pour les résoudre partiel ou complètement.

7 Références

[1] M. E. Celebi, H. A. Kingravi, B. Uddin, H. Iyatomi, Y. A. Aslandogan, W. V. Stoecker, R. H. Moss, "A methodological approach to the classification of dermoscopy images", Comput. Med. Imag. Grap., vol. 31, no. 6, pp. 362-373, 2007.

[2] Gómez, W., Pereira, W. & Infantosi, A. F. C. Analysis of co-occurrence texture statistics as a function of gray-level quantization for classifying breast ultrasound. IEEE Trans. Med. Imag. 31, 1889–1899 (2012).

[3] R. Oliveira, J. Papa, A. Pereira, J. Tavares Computational methods for pigmented skin lesion classification in images : review and future trends Neural Compu. Appl. (2016), pp. 1-24

[4] R. Garnavi, M. Aldeen, J. Bailey, "Computer-aided diagnosis of melanoma using border and wavelet-based texture analysis", IEEE Trans. Inf. Technol. Biomed., vol. 16, no. 6, pp. 1239-1252, Nov. 2012.

Annexes

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Mar  3 15:38:17 2019

@author: Raul Alfredo de Sousa Silva
Features extraction

"""
# Imports (used libraries)

import cv2
import numpy as np
import matplotlib.pyplot as plt
import skimage.morphology as morpho
from skimage.transform import resize
from mpl_toolkits.axes_grid1 import AxesGrid


##############################################################################
# Basic function from ancient TPs of IMA coures
##############################################################################
def Get_values_without_error(im,XX,YY):
    """ retouren une image de la taille de XX et YY
     qui vaut im[XX,YY] mais en faisant attention a ce que XX et YY ne debordent
     pas """
    sh=XX.shape
    defaultval=0;
    if len(im.shape)>2: #color image !
        defaultval=np.asarray([0,0,0])
        sh=[*sh,im.shape[2]]
    imout=np.zeros(sh)
    (ty,tx)=XX.shape[0:2]
    for k in range(ty):
        for l in range(tx):
            posx=int(XX[k,l]-0.5)
            posy=int(YY[k,l]-0.5)
            if posx<0 or posx>=im.shape[1] or posy<0 or posy>=im.shape[0]:
                valtmp=defaultval
            else:
                valtmp=im[posy,posx]
            imout[k,l]=valtmp

    return imout

def rotation(im,theta,alpha=1.0,x0=None,y0=None,ech=0,clip=True):
    """
   %
%Effectue la transformation geometrique d'une image par
%une rotation + homothetie
%
% x' = alpha*cos(theta)*(x-x0) - alpha*sin(theta)*(y-y0) + x0
% y' = alpha*sin(theta)*(x-x0) + alpha*cos(theta)*(y-y0) + y0
%
% theta : angle de rotation en degres
% alpha : facteur d'homothetie (defaut=1)
% x0, y0 : centre de la rotation (defaut=centre de l'image)
% ech : plus proche voisin (defaut=0) ou bilineaire (1)
% clip : format de l'image originale (defaut=True), image complete (False)
%

    """
    dy=im.shape[0]
    dx=im.shape[1]

    if x0 is None:
        x0=dx/2.0
    if y0 is None:
```

```
68          y0=dy/2.0
69      v0=np.asarray([x0,y0]).reshape((2,1))
70      theta=theta/180*np.pi
71      ct=alpha*np.cos(theta)
72      st=alpha*np.sin(theta)
73      matdirect=np.asarray([[ct,-st],[st,ct]])
74      if clip==False:
75          #ON CALCULE exactement la transformee des positions de l'image
76          # on cree un tableau des quatre points extremes
77          tabextreme=np.asarray([[0,0,dx,dx],[0,dy,0,dy]])
78          tabextreme_trans= matdirect@(tabextreme-v0)+v0
79          xmin=np.floor(tabextreme_trans[0].min())
80          xmax=np.ceil(tabextreme_trans[0].max())
81          ymin=np.floor(tabextreme_trans[1].min())
82          ymax=np.ceil(tabextreme_trans[1].max())
83
84      else:
85          xmin=0
86          xmax=dx
87          ymin=0
88          ymax=dy
89      if len(im.shape)>2:
90          shout=(int(ymax-ymin),int(xmax-xmin),im.shape[2]) # image couleur
91      else:
92          shout=(int(ymax-ymin),int(xmax-xmin))
93      dyout=shout[0]
94      dxout=shout[1]
95      eps=0.0001
96      Xout=np.arange(xmin+0.5,xmax-0.5+eps)
97      Xout=np.ones((dyout,1))@Xout.reshape((1,-1))
98
99      Yout=np.arange(ymin+0.5,ymax-0.5+eps)
100     Yout=Yout.reshape((-1,1))@np.ones((1,dxout))
101
102     XY=np.concatenate((Xout.reshape((1,-1)),Yout.reshape((1,-1))),axis=0)
103     XY=np.linalg.inv(matdirect)@(XY-v0)+v0
104     Xout=XY[0,:].reshape(shout)
105     Yout=XY[1,:].reshape(shout)
106     if ech==0: # plus proche voisin
107         out=Get_values_without_error(im,Xout,Yout)
108     else:  #bilineaire
109         assert ech == 1 , "Vous avez choisi un echantillonnage inconnu"
110         Y0=np.floor(Yout-0.5)+0.5 # on va au entier+0.5 inferieur
111         X0=np.floor(Xout-0.5)+0.5
112         Y1=np.ceil(Yout-0.5)+0.5
113         X1=np.ceil(Xout-0.5)+0.5
114         PoidsX=Xout-X0
115         PoidsY=Yout-Y0
116         PoidsX[X0==X1]=1 #points entiers
117         PoidsY[Y0==Y1]=1 #points entiers
118         I00=Get_values_without_error(im,X0,Y0)
119         I01=Get_values_without_error(im,X0,Y1)
120         I10=Get_values_without_error(im,X1,Y0)
121         I11=Get_values_without_error(im,X1,Y1)
122
            out=I00*(1.0-PoidsX)*(1.0-PoidsY)+I01*(1-PoidsX)*PoidsY+I10*PoidsX*(1-PoidsY)+I11
            *PoidsX*PoidsY
123     return out
124
125 def strel(forme,taille,angle=45):
126     """renvoie un element structurant de forme
127      'diamond'  boule de la norme 1 fermee de rayon taille
128      'disk'     boule de la norme 2 fermee de rayon taille
129      'square'   carre de cote taille (il vaut mieux utiliser taille=impair)
130      'line'     segment de langueur taille et d'orientation angle (entre 0 et 180 en
         degres)
131       (Cette fonction n'est pas standard dans python)
```

```
        """

        if forme == 'diamond':
            return morpho.selem.diamond(taille)
        if forme == 'disk':
            return morpho.selem.disk(taille)
        if forme == 'square':
            return morpho.selem.square(taille)
        if forme == 'line':
            angle=int(-np.round(angle))
            angle=angle%180
            angle=np.float32(angle)/180.0*np.pi
            x=int(np.round(np.cos(angle)*taille))
            y=int(np.round(np.sin(angle)*taille))
            if x**2+y**2 == 0:
                if abs(np.cos(angle))>abs(np.sin(angle)):
                    x=int(np.sign(np.cos(angle)))
                    y=0
                else:
                    y=int(np.sign(np.sin(angle)))
                    x=0
            rr,cc=morpho.selem.draw.line(0,0,y,x)
            rr=rr-rr.min()
            cc=cc-cc.min()
            img=np.zeros((rr.max()+1,cc.max()+1) )
            img[rr,cc]=1
            return img
        raise RuntimeError('Erreur dans fonction strel: forme incomprise')

    def extract(image,seg,img_id):
    ################################################################################
        # Color channels - 6 space colors
    ################################################################################
        # Creating channels in the 6 space colors

        #RGB
        R = image[:,:,0]
        G = image[:,:,1]
        B = image[:,:,2]

        #Normalized RGB
        image_n = image.astype(np.uint16)
        r = image_n[:,:,0]/(image_n[:,:,0]+image_n[:,:,1]+image_n[:,:,2]+0.00001)
        g = image_n[:,:,1]/(image_n[:,:,0]+image_n[:,:,1]+image_n[:,:,2]+0.00001)
        b = image_n[:,:,2]/(image_n[:,:,0]+image_n[:,:,1]+image_n[:,:,2]+0.00001)

        #HSV
        image_hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
        h = image_hsv[:,:,0]
        s = image_hsv[:,:,1]
        v = image_hsv[:,:,2]

        #I1/2/3
        I1 = (R+G+B)/3
        I2 = (R+B)/2
        I3 = (2*G-R-B)/4
        #image_I = np.transpose(np.array([I1,I2,I3]))

        #l1/2/3
        l1 = (R-G)**2/((R-G)**2+(R-B)**2+(G-B)**2+.000001)
        l2 = (R-B)**2/((R-G)**2+(R-B)**2+(G-B)**2+.000001)
        l3 = (G-B)**2/((R-G)**2+(R-B)**2+(G-B)**2+.000001)
        #image_l = np.transpose(np.array([l1,l2,l3]))

        # CIE l*a*b*
        image_luv = cv2.cvtColor(image, cv2.COLOR_RGB2Luv)
        l = image_luv[:,:,0]
```

```
    u = image_luv[:,:,1]
    v2 = image_luv[:,:,2]

    channels = [G,B,r,g,b,h,s,v,I1,I2,I3,l1,l2,l3,l,u,v2]
    colors = np.expand_dims(R, axis=2)
    for element in channels:
        colors = np.append(colors, np.expand_dims(element, axis=2),axis=2)
    del channels,R,G,B,r,g,b,h,s,v,I1,I2,I3,l1,l2,l3,l,u,v2
    del image_hsv,image_luv,image_n
################################################################################
    # Shape Features - 13
################################################################################
    [m,n] = np.shape(seg)
    ind = np.where(seg==1)
    row = np.expand_dims(ind[0], axis=1)
    col = np.expand_dims(ind[1], axis=1)

    # First order geometric moments
    m10 = sum(row)
    m01 = sum(col)
    m00 = sum(sum(seg))
    r0 = m10/m00
    c0 = m01/m00

    # Second order central moments
    mu02 = sum((col-c0)**2)
    mu20 = sum((row-r0)**2)
    mu11 = sum((col-c0)*(row-r0))
    m10c = sum(row*colors[ind])
    m01c = sum(col*colors[ind])
    m00c = sum(colors[ind])
    mu02c = sum((col-c0)**2*image[ind])
    mu20c = sum((row-r0)**2*image[ind])
    mu11c = sum((col-c0)*(row-r0)*image[ind])

    # Perimeter
    perimeter = 0
    for k in range(0,len(ind[0])):
        i = ind[0][k]
        j = ind[1][k]
        zeroc = max(0,j-1)
        infic = min(n-1,j+1)
        zeror = max(0,i-1)
        infir = min(m-1,i+1)
        if (seg[i,infic]+seg[i,zeroc]+seg[infir,j]+seg[zeror,j])<4:
            perimeter+=1


    # Aspect ratio
    L1 = (8*(mu02+mu20 + ((mu02-mu20)**2 + 4*mu11)**(1/2) ))**(1/2)
    L2 = (8*(mu02+mu20 - ((mu02-mu20)**2 + 4*mu11)**(1/2) ))**(1/2)
    Ar = L1/L2
    epsilon = ((mu02-mu20)**2 + 4*mu11)/((mu02-mu20)**2)
    theta = 0.5*np.tan(2*mu11/(mu20-mu02))

    mu20 = mu20
    mu02 = mu02
    mu11 = mu11
    # Assimetry 1 and 2
        # Rotation around the centroid of the lesion
    boolean_x = rotation(seg,180*theta[0]/np.pi,x0=round(c0[0]),y0=round(r0[0]))
    boolean_x = boolean_x.astype(np.uint8) # Cast to integer
        # number of pixels that belongs to both images (original an rotated)
    diffx = sum(sum(boolean_x*seg))

    boolean_y = rotation(seg,180*theta[0]/np.pi-90,x0=round(c0[0]),y0=round(r0[0]))
    boolean_y = boolean_y.astype(np.uint8) # Cast to integer
```

```
            # number of pixels that belongs to both images (original an rotated)
        diffy = sum(sum(boolean_y*seg))
        Ax = m00-diffx
        Ay = m00-diffy
        A1 = min(Ax,Ay)/m00
        A2 = (Ax+Ay)/m00

        #Equivalent diameter parametrised by the axis L1 ond (or) L2
        eqd = round(2*np.sqrt(m00/np.pi))
        L1 = L1
        L2 = L2
        # Shape features
        shapes =
        [m00,eqd,perimeter,Ar[0],epsilon[0],theta[0],A1,A2,L1[0],L2[0],mu20[0],mu02[0],mu11[0
        ]]

        # Grouping all features
        metrics = [img_id]
        for element in shapes:
            metrics.append(element)

        del shapes
    ###############################################################################
        # Color features - 348
    ###############################################################################
        se=strel('disk',round(0.05*eqd/2))
        se2=strel('disk',round(0.1*eqd/2))
        # dilatation
        ignore=morpho.dilation(seg,se)
        consider = morpho.dilation(seg,se2)
        inner = consider-ignore
        # erosion
        ignore=morpho.erosion(seg,se)
        consider = morpho.erosion(seg,se2)
        outer = ignore-consider

        seg_ex = np.expand_dims(seg, axis=2)
        inner_ex = np.expand_dims(inner, axis=2)
        outer_ex = np.expand_dims(outer, axis=2)

        col_seg = (seg_ex*colors)
        col_inner = (inner_ex*colors)
        col_outer = (outer_ex*colors)

        # 324 features of mean and standard deviation
        for i in range(0,colors.shape[2]):
            metrics.append(np.mean(col_seg[:,:,i]))
            metrics.append(np.std(col_seg[:,:,i]))
            metrics.append(np.mean(col_inner[:,:,i]))
            metrics.append(np.std(col_inner[:,:,i]))
            metrics.append(np.mean(col_outer[:,:,i]))
            metrics.append(np.std(col_outer[:,:,i]))
            metrics.append(np.mean(col_outer[:,:,i])/np.mean(col_inner[:,:,i]))
            metrics.append(np.std(col_outer[:,:,i])/np.std(col_inner[:,:,i]))
            metrics.append(np.mean(col_outer[:,:,i])/np.mean(col_seg[:,:,i]))
            metrics.append(np.std(col_outer[:,:,i])/np.std(col_seg[:,:,i]))
            metrics.append(np.mean(col_inner[:,:,i])/np.mean(col_seg[:,:,i]))
            metrics.append(np.std(col_inner[:,:,i])/np.std(col_seg[:,:,i]))
            metrics.append(np.mean(col_outer[:,:,i])-np.mean(col_inner[:,:,i]))
            metrics.append(np.std(col_outer[:,:,i])-np.std(col_inner[:,:,i]))
            metrics.append(np.mean(col_outer[:,:,i])-np.mean(col_seg[:,:,i]))
            metrics.append(np.std(col_outer[:,:,i])-np.std(col_seg[:,:,i]))
            metrics.append(np.mean(col_inner[:,:,i])-np.mean(col_seg[:,:,i]))
            metrics.append(np.std(col_inner[:,:,i])-np.std(col_seg[:,:,i]))

        # 6 features of color asymmetry
        thetacol = 0.5*np.tan(2*mu11c/(mu20c-mu02c))
```

```
        # Rotation around the centroid of the lesion
    for i in range(0,3):
        thetac = thetacol[i]
        boolean_x =
        rotation(seg,180*thetac/np.pi,x0=round(m01c[i]/m00),y0=round(m10c[i]/m00))
        boolean_x = boolean_x.astype(np.uint8) # Cast to integer
        diffx = sum(sum(boolean_x*seg))
        boolean_y =
        rotation(seg,180*thetac/np.pi-90,x0=round(m01c[i]/m00),y0=round(m10c[i]/m00))
        boolean_y = boolean_y.astype(np.uint8) # Cast to integer
        diffy = sum(sum(boolean_y*seg))
        Ax = m00-diffx
        Ay = m00-diffy
        A1 = min(Ax,Ay)/m00
        A2 = (Ax+Ay)/m00
        metrics.append(A1)
        metrics.append(A2)

    # 18 features of centroidal distances
    r0c = m10c/m00c
    c0c = m01c/m00c
    dist = (r0-r0c)**2+(c0-c0c)**2
    # Normalizing
    dist = dist/max(dist)
    for i in range(0,len(dist)):
        metrics.append(dist[i])
    del colors
##############################################################################
    # Texture features - 72
##############################################################################
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    #imshow(gray)
    comp = np.array([seg,inner,outer])
    ind = np.where(comp == 1)
    # Requantized image
    ncolor = 64
    img_c = np.round(np.array([ncolor-1])*gray/255).astype(np.uint8)
    # Co-occurence matrix for 0°(0), 45°(1), 90°(2), 135°(3)
    C = np.zeros([64,64,3,4])
    # Size of the masks (to normalization)
    size = np.array([[[sum(sum(seg)),sum(sum(inner)),sum(sum(outer))]]])
    for l in range(0,len(ind[0])):
        k = ind[0][l]
        i = ind[1][l]
        j = ind[2][l]
        maxi = min(n-1,j+1)
        minij = max(0,j-1)
        minii = max(0,i-1)
        C[img_c[i,j],img_c[i,maxi],k,0] +=1
        C[img_c[i,j],img_c[minii,maxi],k,1] +=1
        C[img_c[i,j],img_c[minii,j],k,2] +=1
        C[img_c[i,j],img_c[minii,minij],k,3] +=1

    Cmean = np.mean(C,axis=3)/size

    # Maximum probability
    mp = np.max(Cmean,axis=(0,1))
    # Energy
    E = sum(sum(Cmean**2))
    # Entropy
    S = -sum(sum(Cmean*np.log(Cmean+0.00001)))
    # Auxiliary matrix with |i-j| to each (i,j)
    pos = np.linspace(0,ncolor-1,ncolor)
    m1 = np.matlib.repmat(pos,ncolor,1)
    o = np.ones([ncolor,ncolor])
    mpos = abs(m1 - np.transpose(pos*o))
    mpos = np.expand_dims(mpos, axis=2)
```

```
        m = np.expand_dims(m1, axis=2)
        # Dissimilarity
        D = sum(sum(mpos*Cmean))
        # Contrast
        C = sum(sum(mpos**2*Cmean))
        # Inverse difference
        ID = sum(sum(Cmean/(1+mpos)))
        # Inverse difference moment
        IDM = sum(sum(Cmean/(1+mpos**2)))
        # Auxiliary variables
        muj = sum(sum(m*Cmean))
        mui = sum(sum(np.transpose(m,(1,0,2))*Cmean))
        sigmaj = sum(sum((m-muj)**2*Cmean))
        sigmai = sum(sum((np.transpose(m,(1,0,2))-mui)**2*Cmean))
        # Correlation
        COR = sum(sum((np.transpose(m,(1,0,2))-mui)*(m-muj)*Cmean/(sigmai*sigmaj)))
        # Texture features for seg, inner and outer
        t = np.array([mp,E,S,D,C,ID,IDM,COR])
        del mp,E,S,D,C,ID,IDM,COR,mpos,o,m,pos,m1,Cmean
        # Expanding texture features: same operations made for color (- and /)
        new1 = np.transpose(np.array([t[:,2]-t[:,1],t[:,2]-t[:,0],t[:,1]-t[:,0]]))
        new2 = np.transpose(np.array([t[:,2]/t[:,1],t[:,2]/t[:,0],t[:,1]/t[:,0]]))

        for i in range(0,8):
            for j in range(3):
                metrics.append(t[i,j])
                metrics.append(new1[i,j])
                metrics.append(new2[i,j])
        return metrics

```

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  4 23:23:08 2019

@author: Raul Alfredo de Sousa Silva
"""
import csv
import pandas as pd
import numpy as np
from skimage.io import imread
from skimage.transform import resize
import features_extractor as fs

def features(filename1,filename2,address1,address2,featurefile1,featurefile2,lim=3000,a=0,b=0):
    '''
    Given the images located in the 'filenamex', which names are included in 'addressx', this fuction returns all the
    433 features of your dataset into a file with the name given in 'featurefilex'.
    The features are:
        13 features of shape
        348 features of color
        72 features of texture
    To see more details about this features please refer to the report
    Variables:
    # Change this to access training set images
        filename1 = 'XXX'
    # Change this to access test set images
        filename2 = 'XXX'
    # Change this to access training set names
        address1 = 'XXX'
    # Change this to access test set names
        address2 = 'XXX'
    # Change this to rename the training set features file
        featurefile1 = 'XXX'
    # Change this to rename the test set features file
        featurefile2 = 'XXX'
    # A reduction is applied to images in which one of the dimension is greater
    then 3000 pixels by default (to reduce the computational time). You are able
    to change it.
    # Threshold for reduction
    lim = 3000
    # New dimensions in case of reduction
    a=0 size of rows
    b=0 size of columns
    If they were let at 0 the default reduction will be applied, which means
    each dimension divided by 2.
    Syntax:
        features(filename1,filename2,address1,address2,featurefile1,
                 featurefile2,lim=3000,a=0,b=0)
    Observation1: To reduce computational time images with more than 3000 pixels
    are reduced by 2 in the two dimensions
    Observation3: We suppose to have .jpg images
    Observation2: We suppose to have a segmentation map of the image with
    filename    <image>_segmentation.jpg
    '''
    # Change this to access training set images
    #filename1 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\im\\'
    # Change this to access test set images
    #filename2 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\im\\'
    # Change this to access training set names
    #address1 =
    'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\train.csv'
    # Change this to access test set names
    #address2 =
    'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\test.csv'
```

```
    # Change this to rename the training set file
    #featurefile1 = 'features_train.csv'
    # Change this to rename the test set file
    #featurefile2 = 'features_test.csv'
    #a=0
    #b=0
    #lim = 3000

    # Creating labels
    nfeat = 433
    labels = ["ImageID"]
    for i in range(0,nfeat):
        labels.append('f{}'.format(i))
    # Loading features of the training-set into a .csv file
    df = pd.read_csv(address1)
    X_df = df['ImageId']
    X_train = X_df.values
    # Creating archive and writing labels
    with open(featurefile1, 'w',newline='') as csvfile:
        filewriter = csv.writer(csvfile)
        # Writing labels
        filewriter.writerow(labels)
    # Writing training-set features in the file
    i=0
    for name_im in X_train:
        filename = filename1+'{}.jpg'.format(name_im)
        image = imread(filename)
        filename_Segmentation = filename1+'{}_segmentation.jpg'.format(name_im)
        image_Segmentation = imread(filename_Segmentation) # Value 0 or 255
        # Use if necessary
        (h,w,c) = image.shape
        if (h > lim or w > lim):
            if (a == 0 or b==0):
                a = int(h/2)
                b = int(w/2)
            h_n = a
            w_n = b
            image = resize(image,(h_n,w_n), mode='reflect')
            image_Segmentation = resize(image_Segmentation,(h_n,w_n), mode='reflect')
            # To get uint8
            seg = (np.round(image_Segmentation)).astype(np.uint8)
            image = (np.round(255*image)).astype(np.uint8)
        else:
            seg = (image_Segmentation/255).astype(np.uint8)
        print()
        features = fs.extract(image,seg,name_im)
        with open(featurefile1, 'a',newline='') as csvfile:
            filewriter = csv.writer(csvfile)
            filewriter.writerow(features)
        i+=1
        print(i,"out of 700")

    # Loading features of the trainingset into a .csv file
    df = pd.read_csv(address2)
    X_df = df['ImageId']
    y_df = df['Malignant']
    X_test = X_df.values
    y_test = y_df.values

    # Creating labels

    with open(featurefile2, 'w',newline='') as csvfile:
        filewriter = csv.writer(csvfile)
        # Writing labels
        filewriter.writerow(labels)
    # Writing features of the test-set into a .csv file
    i=0
```

```
130
131         for name_im in X_test:
132             filename = filename2+'{}.jpg'.format(name_im)
133             image = imread(filename)
134             filename_Segmentation = filename2+'{}_segmentation.jpg'.format(name_im)
135             image_Segmentation = imread(filename_Segmentation) # Value 0 or 255
136             # Use if necessary
137             (h,w,c) = image.shape
138             if (h > lim or w > lim):
139                 if (a == 0 or b==0):
140                     a = int(h/2)
141                     b = int(w/2)
142                 h_n = a
143                 w_n = b
144                 image = resize(image,(h_n,w_n), mode='reflect')
145                 image_Segmentation = resize(image_Segmentation,(h_n,w_n), mode='reflect')
146                 # To get uint8
147                 seg = (np.round(image_Segmentation)).astype(np.uint8)
148                 image = (np.round(255*image)).astype(np.uint8)
149             else:
150                 seg = (image_Segmentation/255).astype(np.uint8)
151             features = fs.extract(image,seg,name_im)
152             with open(featurefile2, 'a',newline='') as csvfile:
153                 filewriter = csv.writer(csvfile)
154                 filewriter.writerow(features)
155             i+=1
156             print(i,"out of 300")
157
158
```

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 15 22:20:31 2019

@author: Raul Alfredo de Sousa Silva
"""

#%% Printing versions
print("Library versions used in this work:")
print("csv version: 1.0")
print("numpy version: 1.15.4")
print("Tensorflow version:1.13.1")
print("matplotlib version: 3.0.2")
print("sklearn version: 0.20.1")
#%%
import csv
import itertools
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn import decomposition
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import  cross_val_score, GridSearchCV, KFold

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)

from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)


#%%
# Code from scikit-learn
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="red")

    plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label')
    plt.tight_layout()

class_names = ["healthy","cancer"]
#%% Writing predictions

def generate_file(y,filename):
    with open('features_test.csv', 'r') as csvfile:
        csv_reader = csv.DictReader(csvfile)
        A = []
        for row in csv_reader:
            A.append(row["ImageID"])

    labels = [['ImageId','Malignant']]
    for i in range(0,300):
        labels.append([A[i],int(y[i])])

    with open(filename, 'w',newline='') as csvfile:
        filewriter = csv.writer(csvfile)
        filewriter.writerows(labels)

#%% Reading features

################################################################################
'''
Observation: Features were obtained based on the two other .py files delivered
with this one. It's a very long procedure even if I did all I could to
accelerate it in terms of vectorization of the code. It took me around 17 hours
to finish so I do not recommend you to repeat, but if you really would like to
evaluate it you can uncomment the next lines below to turn it.
If not, just continue, the file with the featured should be attached with this
code too.
'''
#import main_features as mf
## Change this to access training set images
#filename1 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\im\\'
## Change this to access test set images (they can eventually be in different folders)
#filename2 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\im\\'
## Change this to access training set names
#address1 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\train.csv'
## Change this to access test set names
#address2 = 'C:\\Users\\raul-\\Documents\\DataSet-Challenge-IMA205-2019\\data\\test.csv'
## Change this to rename the training set features file
#featurefile1 = 'features_train_t.csv'
## Change this to rename the test set features file
#featurefile2 = 'features_test_t.csv'
#
## A reduction is applied to images in which one of the dimension is greater
## then 3000 pixels by default (to reduce the computational time). You are able
## to change it.
## Threshold for reduction
#lim = 3000
## New dimensions in case of reduction
#a=0 # size of rows
#b=0 # size of columns
## If they were let at 0 the default reduction will be applied, which means
## each dimension divided by 2.
#mf.features(filename1,filename2,address1,address2,featurefile1,featurefile2,
#            lim=3000,a=0,b=0)

################################################################################

with open('features_train.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    row_count = 0
    A = []
```

```
    for row in csv_reader:
        for element in row:
            if line_count ==0:
                row_count+=1
            if row[element].find('IM') < 0:
                A.append(float(row[element]))
        line_count+=1

with open('features_test.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    row_count = 0
    for row in csv_reader:
        for element in row:
            if line_count ==0:
                row_count+=1
            if row[element].find('IM') < 0:
                A.append(float(row[element]))
        line_count+=1


X = np.reshape(A,(1000,(row_count-1)))
#%% Reading classes

with open('train.csv', mode='r') as csv_file:
    csv_reader = csv.DictReader(csv_file)
    line_count = 0
    row_count = 0
    B = []
    for row in csv_reader:
        for element in row:
            if line_count ==0:
                row_count+=1
            if row[element].find('IM') < 0:
                B.append(int(row[element]))
        line_count+=1
y = np.reshape(B,(700))
#%%
##############################################################################
#                                 Preprocessing                              #
##############################################################################

# Scale data (each feature will have average equal to 0 and unit variance)
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)

# Use number of components take explain 95% of variability
pca = decomposition.PCA(n_components=0.95)
pca.fit(X)
X_pca = pca.transform(X)


X_train = X_pca[:700,:]
X_test = X_pca[700:,:]

#%%
##############################################################################
#                               First test: KNN                              #
##############################################################################

# Direct training

print("Fitting K-nearest neighbour to the training set")
p_grid_KNN = {'n_neighbors': [1,3,5,7,9,11,13,15]}
KNN = KNeighborsClassifier()
inner_cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)

grid_KNN = GridSearchCV(estimator=KNN, param_grid=p_grid_KNN,
                        scoring='balanced_accuracy', cv=5)
grid_KNN.fit(X_train, y)
print("Best training Score: {}".format(grid_KNN.best_score_))
print("Best training params: {}".format(grid_KNN.best_params_))

y_pred = grid_KNN.predict(X_test)
name = 'test_KNN.csv'
generate_file(y_pred,name)

#%%
##############################################################################
#                        Applying boosting to knn                            #
##############################################################################
# Creating multiple predictors (21)
p_grid_KNN = {'n_neighbors': [1,3,5,7,9,11,13,15]}
KNN = KNeighborsClassifier()

T = 21
m = len(X_train)
ys = y - (y==0)
p = 1/m*np.ones(m)
index = np.zeros(400)
alpha = np.zeros(T)
grid_KNN = []
S = X_train
for t in range(T):
    Z = sum(p)
    p /= Z
    pcum = np.cumsum(p)
    for i in range(400):
        index[i] = np.where(pcum >= np.random.rand())[0][0]
    index = index.astype(np.int)
    St = X_train[index,:]
    yt = ys[index]
    grid_KNN.append(GridSearchCV(estimator=KNN, param_grid=p_grid_KNN,
                                 scoring='balanced_accuracy', cv=5))
    grid_KNN[-1].fit(St, yt)
    et = sum((grid_KNN[-1].predict(St)!=yt))/len(St)
    alpha[t] = 0.5*np.log((1-et)/(et+1e-10))
    correct1 = (grid_KNN[-1].predict(S) != ys).astype(np.int)
    correct2 = (grid_KNN[-1].predict(S) == ys).astype(np.int)
    correct = correct1 - correct2
    p = p/Z*np.e**(correct*alpha[t])

y_pred = np.zeros(300)
for t in range(T):
    y_pred += alpha[t]*grid_KNN[t].predict(X_test)
y_pred_fin = (y_pred > 0)
y_pred_fin = y_pred_fin.astype(np.int)

name = 'test_KNNboost.csv'
generate_file(y_pred_fin,name)


#%%

##############################################################################
#                              Second test: SVM                              #
##############################################################################

# Fitting Linear SVM on original data
print("Fitting Linear SVM to the training set")

p_grid_lsvm = {'C': [1e-3,1e-2,1e-1,1,2,3,4,5,6,7,8,9,1e1]}
```

```
Lsvm = LinearSVC(class_weight='balanced')
inner_cv = KFold(n_splits=5, shuffle=True, random_state=42)
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)
# Nested CV with parameter optimization
clf = GridSearchCV(estimator=Lsvm, param_grid=p_grid_lsvm, cv=inner_cv)
nested_score = cross_val_score(clf, X_train, y, cv=outer_cv,
                               scoring='balanced_accuracy')

print("Average and std Nested Cv score : {0} +- {1}".format(nested_score.mean(),
      nested_score.std() ))

# Looking for the best hyperparameters
grid_lsvm = GridSearchCV(estimator=Lsvm, param_grid=p_grid_lsvm,
                         scoring='balanced_accuracy', cv=5)
grid_lsvm.fit(X_train, y)
print("Best Score: {}".format(grid_lsvm.best_score_))
print("Best params: {}".format(grid_lsvm.best_params_))

y_pred = grid_lsvm.predict(X_test)
name = 'test_SVM.csv'
generate_file(y_pred,name)

#%%
##############################################################################
#                         Applying non_linearity to SVM                      #
##############################################################################
# Fitting Non-linear SVM
print("Fitting Non-linear SVM to the training set")
p_grid_nlsvm = {'C': [1e-3,1e-2,1e-1,1,2,3,4,5,6,7,8,9,1e1],
                'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
NLsvm = SVC(kernel='rbf', class_weight='balanced')
inner_cv = KFold(n_splits=5, shuffle=True, random_state=42)
outer_cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Looking for the best hyperparameters
grid_nlsvm = GridSearchCV(estimator=NLsvm, param_grid=p_grid_nlsvm,
                          scoring="balanced_accuracy", cv=5)
grid_nlsvm.fit(X_train, y)
print("Best Score: {}".format(grid_nlsvm.best_score_))
print("Best params: {}".format(grid_nlsvm.best_params_))


y_pred = grid_nlsvm.predict(X_test)
name = 'test_SVM_nl.csv'
generate_file(y_pred,name)
#%%
##############################################################################
#                        Third test: Decision Trees                          #
##############################################################################
# Fitting Decision Trees
Tree = DecisionTreeClassifier()
Tree.fit(X_train,y)
# Score in the training set
print('Score in the training set is {0}'.format(Tree.score(X_train,y)) )

y_pred = Tree.predict(X_test)
name = 'test_tree.csv'
generate_file(y_pred,name)

#%%
##############################################################################
#                                Fourth test: MLP                            #
##############################################################################
y_train = tf.keras.utils.to_categorical(y)

def init_weights_and_biases(shape, stddev=0.1, seed_in=None):
    """
```

```
    This function should return Tensorflow Variables containing the initialised
    weights and biases of the network,
    using a normal distribution for the initialisation, with stddev of the
    normal as an input argument

    Parameters
    ----------
    shape : tuple, (n_input_features,n_hidden_1, n_hidden_2,n_classes)
        sizes necessary for defining the weights and biases

    Returns
    -------
    w1, b1, w2, b2, w3, b3: initialised weights and biases, with correct shapes
    """

    # BEGIN STUDENT CODE
    w1 = tf.Variable(tf.random_normal([shape[0],shape[1]], stddev=stddev,
                                      seed=seed_in))
    w2 = tf.Variable(tf.random_normal([shape[1],shape[2]], stddev=stddev,
                                      seed=seed_in))
    w3 = tf.Variable(tf.random_normal([shape[2],shape[3]], stddev=stddev,
                                      seed=seed_in))
    b1 = tf.Variable(tf.zeros([shape[1]]))
    b2 = tf.Variable(tf.zeros([shape[2]]))
    b3 = tf.Variable(tf.zeros([shape[3]]))
    # END STUDENT CODE
    return w1, b1, w2, b2, w3, b3

def forward_prop_multi_layer(X, w1, b1, w2, b2, w3, b3):
    """
    This function should define the network architecture, explained above

    Parameters
    ----------
    X : input to network
    w1, w2, w3 : Tensorflow Variables
        network weights
    b1, b2, b3 : Tensorflow Variables
        network biases

    Returns
    -------
    Y_pred :
    the output layer of the network, the classification prediction
    """

    # BEGIN STUDENT CODE
    A1 = tf.nn.sigmoid(tf.matmul(X, w1)+b1)
    A2 = tf.nn.softmax(tf.matmul(A1,w2)+b2)
    Y_pred = tf.nn.softmax(tf.matmul(A2,w3)+b3)
    # END STUDENT CODE
    return Y_pred

def accuracy(Y_pred, Y_true):
    """
    This function calculates the network's accuracy, ie the percentage of
    correct classifications.
    Here, we consider the class with the highest score in the network's output

    Parameters
    ----------
    Y_pred : Tensorfow variable
        predicted classification of network
    Y_true : Tensorfow variable
        true classes of input

    """
```

```
403
404         acc = tf.equal(tf.argmax(Y_pred, 1), tf.argmax(Y_true, 1))
405         acc = tf.reduce_mean(tf.cast(acc, tf.float32))
406
407         return acc
408
409     RANDOM_SEED = 52#42#
410     tf.set_random_seed(RANDOM_SEED)
411
412     # Network Parameters
413     n_hidden_1 = 256 # 1st layer number of neurons
414     n_hidden_2 = 256 # 2nd layer number of neurons
415     n_input = X_train.shape[1]
416     n_classes = y_train.shape[1] # MNIST total classes (0-9 digits)
417
418     # tf Graph input
419     X_input = tf.placeholder("float", [None, n_input])
420     Y_true = tf.placeholder("float", [None, n_classes])
421
422     # Store layers weight & bias
423     stddev = 0.1
424
425     w1,b1,w2,b2,w3,b3 = init_weights_and_biases([n_input, n_hidden_1, n_hidden_2,
426                                     n_classes], stddev=0.1, seed_in=RANDOM_SEED)
427
428     # Construct model
429     Y_pred = forward_prop_multi_layer(X_input,w1,b1,w2,b2,w3,b3)
430
431     # Define loss and optimizer
432     cross_entropy = -tf.reduce_sum(Y_true * tf.log(Y_pred),axis=1)
433
434     loss = tf.reduce_mean(cross_entropy)
435     acc = accuracy(Y_pred, Y_true)
436
437     learning_rate = 0.001
438     optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
439     training_variables = optimizer.minimize(loss)
440
441     # Parameters
442     n_epochs = 500
443     train_accuracy = []
444     test_accuracy = []
445     batch_size = 700
446     display_step = 1
447     n_batches = int(np.ceil(X_train.shape[0]/batch_size))
448     print(n_batches)
449
450
451     with tf.Session() as sess:
452         # Initializing the variables
453         init = tf.global_variables_initializer()
454         sess.run(init)
455
456         for epoch in range(n_epochs):
457             # Loop over all batches
458             for batch_idx in range(n_batches):
459                 #get the next batch in the MNIST dataset and carry out training
460                 poss = batch_size*batch_idx
461                 posf = batch_size*(batch_idx+1)
462
463                 #BEGIN STUDENT CODE
464                 sess.run([training_variables,loss], feed_dict={X_input:
465                     X_train[poss:posf,:], Y_true: y_train[poss:posf]})
466                 #END STUDENT CODE
467
468             # calculate accuracy for this epoch
469             train_accuracy.append(sess.run(acc, feed_dict={X_input: X_train,
```

```
                                                        Y_true:y_train}))
        u = sess.run([Y_pred], feed_dict={X_input: X_test})
        print(".", end='')

print("Training finished")


ypred = np.array(u)
ypred = ypred.reshape(300,2)
ypred = ypred>0.5
ypred = ypred[:,1]
print(sum(ypred.astype(np.int)))
name = 'test_MLP.csv'
generate_file(ypred,name)
```