PEC 3: Clasificación

Presentación

El objetivo de esta prueba de evaluación es aplicar técnicas de clasificación automática a un conjunto de datos de mamografías para el cribaje de cáncer de mama.

Competencias

En este enunciado se trabajan en un determinado grado las siguientes competencias general de máster:

- Capacidad para proyectar, calcular y diseñar productos, procesos e instalaciones en todos los ámbitos de la ingeniería en informática.
- Capacidad para el modelado matemático, cálculo y simulación en centros tecnológicos y de ingeniería de empresa, particularmente en tareas de investigación, desarrollo e innovación en todos los ámbitos relacionados con la ingeniería en informática.
- Capacidad para la aplicación de los conocimientos adquiridos y para solucionar problemas en entornos nuevos o poco conocidos dentro de contextos más amplios y multidisciplinares, siendo capaces de integrar estos conocimientos.
- Poseer habilidades para el aprendizaje continuado, autodirigido y autónomo.
- Capacidad para modelar, diseñar, definir la arquitectura, implantar, gestionar, operar, administrar y mantener aplicaciones, redes, sistemas, servicios y contenidos informáticos.
- Capacidad para asegurar, gestionar, auditar y certificar la calidad de los desarrollos, procesos, sistemas, servicios, aplicaciones y productos informáticos.

Las competencias específicas de esta asignatura que es trabajan son:

- Entender que es el aprendizaje automático en el contexto de la Inteligencia Artificial.
- Distinguir entre los diferentes tipos y métodos de aprendizaje.
- Aplicar las técnicas estudiadas en un caso concreto.

Objetivos

En este PEC se aplicarán a un caso concreto los conceptos del temario sobre clasificación (Tema 4).

Descripción de la PEC a realizar

Datos

El conjunto de datos consta de 516 casos en los que la lesión resultó ser benigna y 445 casos en los que la lesión resultó ser maligna. Cada entrada del registro incluído en el fichero de datos *mamografias.data* contiene los siguientes atributos:

1. Evaluación clínica: del 1 al 5 (1= claramente benigno, 5 = altamente sospechoso de ser maligno).

2. Edad: edad del paciente en años.

3. Forma de la lesión (1 = redondo, 2 = ovalado, 3 = lobular, 4 = irregular).

4. Contorno de la lesión (1 = circunscrito, 2 = microlobulado, 3 = oscurecido, 4 = mal definido, 5 = espiculado).

5. Densidad de la lesión (1 = alta densidad, 2 = iso, 3 = baja densidad, 4 = contiene grasa).

6. Gravedad real de la lesión (0 = benigno, 1 = maligno). Este dato indica la clase real de pertenencia de la lesión.

Un valor de interrogante '?' Corresponde a un valor ausente. El archivo adjunto *mamografias_info.txt* describe la información de dichos atributos. Los archivos pertenecen a la base de datos 'Mammographic Mass Data Set' del Machine Learning Repository de la Universidad de California, Irvine:

http://archive.ics.uci.edu/ml/datasets/Mammographic+Mass

Ejercicio 1

Divida el archivo adjunto en dos: uno para el training y otro para el test. Aplique los programas realizados en los ejercicios 1 (preprocesado) y 2 (descomposición PCA) de la PEC2 a los nuevos conjuntos. Como proyectamos los datos del test de forma coherente?.

Para generar los conjuntos de datos de entrenamiento y de testeo, dividiremos el conjunto de datos inicial en dos partes de igual tamaño. Para evitar que los conjuntos resultantes contengan correlaciones derivadas del orden de los datos en el conjunto inicial, seleccionaremos las observaciones de forma aleatoria utilizando la instrucción 'suffle' de la librería 'random'. La variable 'classes' contiene la etiqueta de clasificación que corresponde a la sexta columna de los datos, mientras que los atributos se almacenan en la variable llt (columnas de la primera a la quinta). A los valores absentes se les asigna la media del atributo (función 'valorAbsent'). La normalización de los datos se realiza mediante una estandarización en la que la media y desviación estándar se calculan a partir del conjunto de test. Hay que tener en cuenta que al haber definido los conjuntos ens de training y test de forma aleatoria, los resultados que obtendremos cada vez que ejecutemos el código serán ligeramente diferentes. Tras diagonalizar la matriz de covarianza de los datos de training, comprobamos que son necesarios cuatro componentes PCA para explicar una varianza del 94% de los datos, por lo que realizamos una proyección de los datos en un espacio PCA de dimensión 4.

```
## Ejercicio 1 ###

from collections import Counter
from random import shuffle
import numpy

# declaracion de funciones:
def valorAbsent (l):
   cl = Counter([l[i] for i in range(len(l)) if (l[i] != '?')])
   moda = float(cl.most_common()[0][0])
   return list(map(lambda x: moda if x=='?' else float(x), l))

#cargar archivo de datos:
ll = list(map(lambda l: (l.strip()).split(','),
            open('mamografias.data', 'r').readlines()))

# unzipear la lista para obtener las clases:
llt = list(zip(*ll))
# classes
classes = numpy.array(llt[5])

# Separar training / test de forma aleatoria:
ind_cl0 = [i for i in range(len(classes)) if classes[i]=='0']
shuffle(ind_cl0)
ind_cl1 = [i for i in range(len(classes)) if classes[i]=='1']
shuffle(ind_cl1)

ind_train_cl0 = ind_cl0[:int(len(ind_cl0)/2)]
ind_test_cl0 = ind_cl0[int(len(ind_cl0)/2):]
ind_train_cl1 = ind_cl1[:int(len(ind_cl1)/2)]
ind_test_cl1 = ind_cl1[int(len(ind_cl1)/2):]

# atributos:
llt = numpy.array(llt[0:5])

# Tratamiento de valores ausentes:
atrs = numpy.array(list(map(valorAbsent, llt)))

# normalización:
prom = list(map(numpy.average,
            atrs[:, ind_train_cl0+ind_train_cl1]))
stds = list(map(numpy.std, atrs[:,ind_train_cl0+ind_train_cl1]))

atrsN = numpy.array([list(map(lambda x: (x - prom[i]) / stds[i], atrs[i]))
        for i in range(len(atrs))])

# Proyección PCA de los datos:
from numpy import *
import pylab
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plot
set_printoptions(precision = 3)

# matriz de covarianza
dades = numpy.array(atrsN).transpose()
dades1 = dades - dades.mean(0)
matcov = dot(dades1.transpose(), dades1)
# vaps i veps
valp1,vecp1 = linalg.eig(matcov)
# ordre
ind_cre = argsort(valp1)
ind_decre = ind_cre[::-1]
val_decre = valp1[ind_decre]
vec_decre = vecp1[:,ind_decre]

n = val_decre / val_decre.sum()
% >>> n[0:4].sum()
% 0.94367012501926562

# Proyeccion de los datos en espacio PCA 4D (94% de la varianza):
d_PCA = [[dot(dades1[i,:],vecp1[:,j])
         for i in range(dades.shape[0])]
        for j in range(4)]
atrsPCA = numpy.array(d_PCA)
```

Ejercicio 2

Aplique a los archivos resultantes del ejercicio 1 las técnicas de clasificación siguientes: KNN (código 4.2), el clasificador lineal basado en distancias (código 4.3) y las SVM con kernel lineal (utilizar, por ejemplo la implementación incluida en las librerias libsvm (código 4.13) que tambien están incluidas en el paquete scikit-learn). Compare los resultados de las tres técnicas de clasificación. Comente si observa ldiferencias significativas entre aplicar las técnicas de clasificación a los datos o a su proyección PCA. ¿Qué conclusiones se derivan?.

Los datos normalizados y los proyectados en un espacio PCA de 4 dimensiones se utilizan ahora para aplicar diferentes técnicas de clasificación automática. El método kNN se implementa de la siguiente forma, siguiendo el código 4.2 de los materiales de la asignatura:

```
############
### kNN ###
############

def deuclidea(x, y):
  return sum(map(lambda a, b: (float(a) - float(b)) ** 2.0,
              x, y)) ** 0.5

def contar(l):
  p = {}
  for x in l:
    p.setdefault(x, 0)
    p[x] += 1
  return p

def kNN(t, k=1):
  ds = list(map(deuclidea, train, [t for x in range(len(train))]))
  kcl = contar([sorted([(ds[i], classesTrain[i])
                 for i in range(len(train))],
                key=lambda x: x[0])[i][1]
            for i in range(k)])
  return max([(x , kcl[x]) for x in kcl.keys()],
          key=lambda x: x[1])[0]

# Classificacio normalitzat
train = atrsN[:,ind_train_cl0+ind_train_cl1].transpose()
test = atrsN[:,ind_test_cl0+ind_test_cl1].transpose()
classesTrain = classes[ind_train_cl0+ind_train_cl1]
classesTest = classes[ind_test_cl0+ind_test_cl1]
prediccions = list(map(kNN, test))

print 'kNN'
print '  Normalitzats:'
print '    Prec.:', float(sum(prediccions == classesTest)) / float(len(prediccions)) * 100, '%'

# Classificacio PCA
train = atrsPCA[:,ind_train_cl0+ind_train_cl1].transpose()
test2 = atrsPCA[:,ind_test_cl0+ind_test_cl1].transpose()
prediccions = list(map(kNN, test2))

print '  PCA:'
print '    Prec.:', float(sum(prediccions == classesTest)) / float(len(prediccions)) * 100, '%'
```

El clasificador lineal basado en distancias se implementa de la siguiente forma, siguiendo el código 4.3 de los materiales:

```
############################
### Classificador lineal ###
############################
from functools import reduce

# declaracion de funciones:
def deuclideaCL(x, y):
  return sum(map(lambda a, b: (float(a) - float(b)) ** 2.0,
                 x[1], y)) ** 0.5

def classifyCL(t):
  ds = list(map(deuclideaCL, centroides,
                [t for x in range(len(centroides))]))
  return min([(ds[i], centroides[i][0])
              for i in range(len(centroides))],
             key=lambda x: x[0])[1]

def calcularCentroides(classe, train):
  filt = list(filter(lambda x: x[0] == x[1],
                     [(classesTrain[i], classe, train[i])
                      for i in range(len(train))]))
  transp = list(zip(*[filt[i][2] for i in range(len(filt))]))
  return (classe,
          list(map(lambda l: reduce(lambda a, b:
                                    float(a) + float(b),
                                    l) / classes[classe], transp)))

# Entrenamiento:
classes = contar(classesTrain)
train = atrsN[:,ind_train_cl0+ind_train_cl1].transpose()
centroides = [calcularCentroides(c, train) for c in classes.keys()]

# Clasificacion:
prediccions = list(map(classifyCL, test))

# Correctes
print 'CL'
print '  Normalizados:'
print '    Prec.:', float(sum(prediccions == classesTest)) / float(len(prediccions)) * 100, '%'

# Entrenamiento:
train2 = atrsPCA[:,ind_train_cl0+ind_train_cl1].transpose()
centroides = [calcularCentroides(c, train2) for c in classes.keys()]

# Clasificacion:
test2 = atrsPCA[:,ind_test_cl0+ind_test_cl1].transpose()
prediccions = list(map(classifyCL, test2))

# Correctas
print '  PCA:'
print '    Prec.:', float(sum(prediccions == classesTest)) / float(len(prediccions)) * 100, '%'
```

El siguiente código implementa un SVM a partir de las librerias librerias libsvm tal y como se describe en el código 4.13 de los materiales. Hay que tener en cuenta los flags de opciones que permiten escoger cada una de las variantes del método, y que se listan al principio del codigo (se utilizan las variables de training y test definidas en los apartados anteriores):

```
##############################
# Clasificador SVM:
##############################

# FUNCION svmtrain de la libreria svmllib: OPCIONES:
##-s svm_type:
##set type of SVM (default 0)
##0 - C-SVC
##1 - nu-SVC
##2 - one-class SVM
##3 - epsilon SVR
##4 - nu-SVR
##-t kernel_type:
##set type of kernel function (default 2)
##0 - linear - u'*v
##1 - polynomial - gamma*u'*v + coef0)^degree
##2 - radial basis function - exp(-gamma*|u-v|^2)
##3 - sigmoid - tanh(gamma*u'*v + coef0)
##4 - precomputed kernel (kernel values in training_set_file)
##-d degree:
##set degree in kernel function (default 3)
##-g gamma:
##set gamma in kernel function (default 1/k). The k means the number of attributes in the input data
##-r coef0:
##set coef0 in kernel function (default 0)
##-c cost:
##set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

## Kernel Lineal:

print 'SVM LINEAL C=1'

list_train = list(map(list,train))

list_test = list(map(list,test))

m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 0 -c 1')

print '  Normalizadas:'

p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)

list_train2 = list(map(list,train2))

list_test2 = list(map(list,test2))

m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 0 -c 1')

print '  PCA:'

p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)
```

Al aplicar los tres códigos anteriores a ambos conjuntos de datos (normalizados y PCA 4D) se obtienen los siguientes resultados, que obviamente variarán ligeramente cada vez que se ejecute el programa puesto que la definición de los conjuntos de training y set se ha realizado de forma aleatoria en el ejercicio 1:

1. Técnica kNN:

Normalizados:

Prec.: 46.3617463617 %

PCA:

Prec.: 46.3617463617 %

2. Clasificador Lineal:

Normalizados:

Prec.: 82.9521829522 %

PCA:

Prec.: 82.7442827443 %

3. SVM LINEAL C=1

Normalizados:

Accuracy = 85.447% (411/481) (classification)

PCA:

Accuracy = 84.1996% (405/481) (classification)

Comprobamos que los resultados obtenidos no difieren significativamente al utilizar los datos normalizados o su proyección PCA, de hecho con los datos normalizados obtenemos 6 casos más correctamente clasificados del conjunto de datos de test que con los datos PCA. Tambien observamos que la técnica SVM es con la que se obtiene un mejor resultado de la clasificación de los datos de test, y que en este caso el clasificador kNN presenta unos resultados excesivamente bajos, probablemente debido a la distribución estadística de los valores en los datos.

Ejercicio 3

Aplique las SVMs con kernel lineal, polinómico y radial con diferentes grados (en el caso de los polinomios) y gamas (en el caso de los radiales) y parámetro C (margen flexible) a los mismos conjuntos que en el ejercicio anterior. Puede seguir utilizando la implementación de SVM de svmlib o la de scikit-learn. ¿Qué conclusiones se pueden extraer sobre los resultados obtenidos con diferentes kernels de SVM?.

En este apartado solamente tenemos que adaptar el código que implementa la técnica SVM del ejercicio anterior para aplicar diferentes Kernels. Para ello hay que seleccionar las diferentes opciones que nos permiten las librerias svmlib en el momento en el que se realiza la fase de entrenamiento de la SVM (llamada a la función svm_train). En nuestro caso hemos considerado varios valores de los parámetros de cada técnica, por lo que se aplicará SVM con los siguientes Kernel:

Kernel LINEAL con C = 1, 5

Kernel POLINOMICO C=1 y GRADO=3, 6

Kernel POLINOMICO C=5 y GRADO=3, 6

Kernel RADIAL C=1 GAMMA=1/4,1

Kernel RADIAL C=5 GAMMA=1/4,1

El código completo se adjunta a continuación:

```
from svmutil import *
## Kernel Lineal:
print 'SVM LINEAL C=1'

list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 0 -c 1')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 0 -c 1')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)
```

```
print 'SVM LINEAL C=5'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 0 -c 5')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 0 -c 5')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

## Kernel Polinomico:
print 'SVM POLINOMICO C=1 GRADO=3'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 1 -d 3')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 1 -d 3')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

print 'SVM POLINOMICO C=1 GRADO=6'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 1 -d 6')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 1 -d 6')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

print 'SVM POLINOMICO C=5 GRADO=3'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-c 5 -t 1 -d 3')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
```

```
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-c 5 -t 1 -d 3')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

print 'SVM POLINOMICO C=5 GRADO=6'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-c 5 -t 1 -d 6')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-c 5 -t 1 -d 6')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)


## Kernel Radial:
print 'SVM RADIAL C=1 GAMMA=1/4 (default)'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 2')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 2')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

print 'SVM RADIAL C=1 GAMMA= 1'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-t 2 -g 1')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-t 2 -g 1')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)
```

```
print 'SVM RADIAL C=5 GAMMA=1/4'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-c 5 -t 2')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-c 5 -t 2')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)

print 'SVM RADIAL C=5 GAMMA=1'
list_train = list(map(list,train))
list_test = list(map(list,test))
m = svm_train(list(numpy.double(classesTrain)), list_train,'-c 5 -t 2 -g 1')
print '  Normalizadas:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test, m)
list_train2 = list(map(list,train2))
list_test2 = list(map(list,test2))
m = svm_train(list(numpy.double(classesTrain)), list_train2,'-c 5 -t 2 -d 1')
print '  PCA:'
p_label, p_acc, p_val = svm_predict(numpy.double(classesTest), list_test2, m)
```

y los resultados obtenidos en cada caso son los siguientes:

SVM LINEAL C=1
  Normalizadas:
Accuracy = 85.447% (411/481) (classification)
  PCA:
Accuracy = 84.1996% (405/481) (classification)
SVM LINEAL C=5
  Normalizadas:
Accuracy = 83.1601% (400/481) (classification)
  PCA:
Accuracy = 83.9917% (404/481) (classification)
SVM POLINOMICO C=1 GRADO=3
  Normalizadas:
Accuracy = 83.5759% (402/481) (classification)

PCA:

Accuracy = 84.6154% (407/481) (classification)

SVM POLINOMICO C=1 GRADO=6

Normalizadas:

Accuracy = 69.6466% (335/481) (classification)

PCA:

Accuracy = 59.8753% (288/481) (classification)

SVM POLINOMICO C=5 GRADO=3

Normalizadas:

Accuracy = 83.5759% (402/481) (classification)

PCA:

Accuracy = 83.7838% (403/481) (classification)

SVM POLINOMICO C=5 GRADO=6

Normalizadas:

Accuracy = 71.1019% (342/481) (classification)

PCA:

Accuracy = 60.499% (291/481) (classification)

SVM RADIAL C=1 GAMMA=1/4 (default)

Normalizadas:

Accuracy = 83.7838% (403/481) (classification)

PCA:

Accuracy = 84.6154% (407/481) (classification)

SVM RADIAL C=1 GAMMA= 1

Normalizadas:

Accuracy = 82.7443% (398/481) (classification)

PCA:

Accuracy = 83.5759% (402/481) (classification)

SVM RADIAL C=5 GAMMA=1/4

Normalizadas:

Accuracy = 83.5759% (402/481) (classification)

PCA:

Accuracy = 84.1996% (405/481) (classification)

SVM RADIAL C=5 GAMMA=1

Normalizadas:

Accuracy = 82.1206% (395/481) (classification)

PCA:

Accuracy = 84.1996% (405/481) (classification)

Podemos comprobar que la mayoría de Kernel escogidos presentan resultados en torno al 80% de aciertos de clasificación, con resultados similares al utilizar datos normalizados o PCA. De hecho, ninguno de los Kernel escogidos permite obtener un rendimiento que sea significativamente superior al que ya presentaba el SVM con Kernel lineal del ejercicio anterior, por lo que esta técnica sería la más adecuada para trabajar con estos datos de entre todas las herramientas de clasificación que se han analizado en este proyecto.

Recursos

Este PEC requiere de los siguientes recursos:

Básicos: Ficheros de datos adjuntos al enunciado.

Complementarios: Manual de teoría de la asignatura, y listados de código del capítulo 4 (Clasificación). Implementación de la técnica SVM incluida en las librerias scikit-learn: http://scikit-learn.org/stable/

Criterios de valoración

Los ejercicios tendrán la siguiente valoración asociada:

Ejercicio 1: 2 punto

Ejercicio 2: 4 puntos

Ejercicio 3: 4 puntos

Es necesario razonar las respuestas en todos los ejercicios. Las respuestas sin justificación no recibirán puntuación.

Formato y fecha de entrega

La PEC debe entregarse antes del **próximo 16 de Mayo** (antes de las 24h).

La solución a entregar consiste en un informe en formato PDF (formato libre) más los archivos de código (*. Py) que usó para resolver la prueba. Estos archivos deben comprimir en un archivo ZIP.

Adjuntar el fichero a un mensaje en el apartado de **Entrega y Registro de AC (RAC)**. El nombre del archivo debe ser ApellidosNombre_IA_PEC2 con la extensión. zip.

Para dudas y aclaraciones sobre el enunciado, diríjase al consultor responsable de su aula.

Nota: **Propiedad intelectual**

A menudo es inevitable, al producir una obra multimedia, hacer uso de recursos creados por terceras personas. Es por tanto comprensible hacerlo en el marco de una práctica de los estudios del Máster de Informática, siempre que esto se documente claramente y no suponga plagio en la práctica.

Por lo tanto, al presentar una práctica que haga uso de recursos ajenos, se presentará junto con ella un documento en el que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y el su estatus legal: si la obra está protegida por copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL ...). El estudiante deberá asegurarse de que la licencia que sea no impide específicamente su uso en el marco de la práctica. En caso de no encontrar la información correspondiente deberá asumir que la obra está protegida por copyright.

Deberán, además, adjuntar los archivos originales cuando las obras utilizadas sean digitales, y su código fuente si corresponde.