Introducció

El present informe exposa els procediments i resultats d'execució de la primera pràctica del curs d'Intel·ligència Artificial Avançada, relativa a Agrupaments i Recomanadors.

El conjunt de dades que s'ha treballat fa referència al comportament dels usuaris de diferents llocs web, i inclou dades d'ús com el nombre de pàgines visitades, el nombre de compres fetes o els *likes* de l'usuari al web. L'objectiu de la pràctica és obtenir patrons de comportament d'aquests usuaris a partir de les dades, que ens permetin recomanar-los webs adients als seus interessos.

El fitxer de dades originals serà comú per a tots els apartats de la pràctica, i té la següent estructura:

Identificador del lloc web	Identificador de l'usuari	Nombre de pàgines vistes	Minuts que ha passat l'usuari al web	Nombre de *likes* de l'usuari al web	Nombre de compres realitzades per l'usuari al web	Nombre d'opinions escrites per l'usuari al web
Enter (1 - 30)	Enter (1 - 100)	Enter (1 - 30)	Enter (1 - 100)	Enter (1 - 50)	Enter (1 - 10)	Enter (1 - 10)

Activitat 1

Com a pas previ a l'anàlisi de dades, realitzarem un tractament previ que asseguri la independència de la contribució de la variable respecte al sistema utilitzat per a mesurar-la. D'altra manera, variables propenses a arribar a rangs alts de valors, com el nombre de pàgines visitades dins d'un web, tindrien major pes en el càlcul de distàncies que d'altres com el nombre de comandes. La estandardització de les dades ens permetrà resoldre aquest problema igualant els rangs i/o la distribució de les dades.

Amb ànims d'explorar diferents mètodes d'estadarització, evaluarem els resultats obtinguts mitjançant les dues tècniques següents:

1. **Escalat 0-1**: cada esdeveniment de cada variable del conjunt de dades es recalcula com:

$$X' = \frac{X - 1}{\max(X) - 1}$$

   Aquest mètode permet que les variables tinguin diferents distribucions però assegura la igualtat de rang.

2. **Estandarització mitjançant la distribució normal tipificada (Z-score):** cada esdeveniment de cada variable del conjunt de dades es recalcula com:

$$Z = \frac{X - \mu}{\sigma}$$

   Aquest mètode assegura que totes les variables tenen una distribució normal amb mitjana 0 i desviació tipus 1.

Les figures següents mostren la distribució de valors de les 5 variables analitzades i els valors límit del rang de cada dimensió de l'arxiu *webs.data* original.

	pageViews	timeOnPage	likes	orders	opinions
MIN	3	8	4	1	1
MAX	30	100	50	10	10

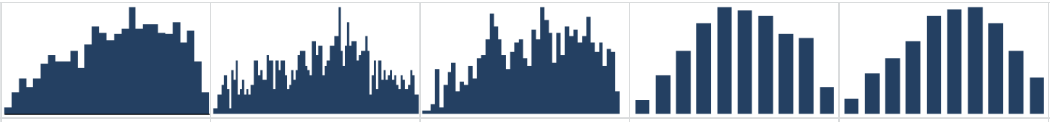
L'escalat 0-1 s'ha realitzat mitjançant la funció *scaleRatings*, recollida a l'arxiu *FunctionsForActivityOne.py* i que es presenta a continuació:

```
# Vector with the maximum values for each topic
MAX_VALUATIONS = [30.0, 100.0, 50.0, 10.0, 10.0, 0]

def scaleVals(vals):
    # Auxiliary function that receives a list of valuations as read
    # from the file and returns it scaled to 0..1
    return [(float(vals[i])-1)/(MAX_VALUATIONS[i]-1) for i in range(len(vals))]

def scaleRatings(array):
    newArray = []
    for l in array:
        tmp = []
        tmp.append(int(l[0])) # Append idWeb
        tmp.append(int(l[1])) # Append idUser
        values = scaleVals(l[2:])
        for x in values:
            tmp.append(x)
        newArray.append(tmp)
    return newArray
```

Per tal de simplificar el treball amb les dades tractades, s'ha definit una nova funció *writeStRatings* recollida a l'arxiu *FunctionsForActivityOne.py* que desa les dades corregides en un nou arxiu *newScWeb.data*:

```
def writeStRatings(array, name="output.data"):
    # AWrite the data
    with open(name, "w") as fp:
        a = csv.writer(fp, delimiter="\t")
        a.writerows(array)
        msg = "Data succesfully written in file " + name
        return msg
```

Les figures següents mostren les distribucions de les dades en el nou arxiu newScWeb.data. Com es pot comprobar, s'ha corregit la diferència de rang entre les dimensions.



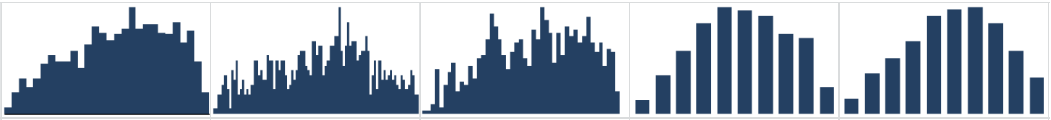
	pageViews	timeOnPage	likes	orders	opinions
MIN	0,068	0,07	0,06	0	1
MAX	1	1	1	1	1

La estandarització, d'altra banda, s'ha realitzat mitjançant la funció *standarizeRatings* recollida a l'arxiu *FunctionsForActivityOne.py* i presentada a continuació:

```python
def mean(lst):
    # Calculates mean for lst
    return sum(lst) / len(lst)

def stddev(lst):
    # Returns the standard deviation of lst
    mn = mean(lst)
    variance = sum([(e-mn)**2 for e in lst])
    return sqrt(variance)

def readRatings(filename="webs.data"):
    # Reads CSV to load a list of lists with every cell value
    lines = [(l.strip()).split("\t") for l in (open(filename).readlines())]
    return lines
    # It will be used in the next activity too

def standarizeRatings(array):
    # The result of standardization (or Z-score normalization) is that
    # the features will be rescaled so that they'll have the properties of a
    # standard normal distribution with mu = 0 and sigma = 1
    aLength = len(array[0])
    rawData = {x: [] for x in range(2, aLength)}
    means = []
    stDevs = []
    for l in array:
        for x in range(2, aLength):
            rawData[x].append(float(l[x]))
    for d in rawData:
        means.append(mean(rawData[d]))
        stDevs.append(stddev(rawData[d]))
    newArray = []
    for l in array:
        tmp = []
        tmp.append(int(l[0])) # Append idWeb
        tmp.append(int(l[1])) # Append idUser
        for x in range(2, aLength):
            if stDevs[x - 2] != 0:
                value = (float(l[x]) - means[x - 2]) / stDevs[x - 2]
                tmp.append(value)
            else:
                value = 0
                tmp.append(means[x - 2])
        newArray.append(tmp)
    return newArray
```

Les figures següents mostren les distribucions de les dades en el nou arxiu *newStWeb.data*.

Com es pot comprovar, a més de la diferència de rang s'ha corregit la distribució de les variables.



	pageViews	timeOnPage	likes	orders	opinions
MIN	-0,08	-0,07	-0,08	-0,07	-0,07
MAX	0,06	0,06	0,06	0,06	0,06

El següent codi (*ActivityOneDef.py*) recull la crida a tots els mètodes esmentats. Tant els arxius de funcions com els nous arxius *newScWebs.data* i *newStWebs.data* estan continguts a l'arxiu comprimit que acompanya aquest informe.

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

# Activity 1: Data Standarization

import FunctionsForActivityOne as f # Import all needed functions

# First, we read the data in webs.data
ratings = f.readRatings("webs.data")
# Now, we standarize the distribution of every variable
standarizedRatings = f.standarizeRatings(ratings)
# The last step is to write the data
msg = f.writeStRatings(standarizedRatings, "newStWebs.data")
print(msg)
# Additionally, we will try with another scaling function to compare results
scaledRatings = f.scaleRatings(ratings)
msg = f.writeStRatings(scaledRatings, "newScWebs.data")
print(msg)
```

Activitat 2

En aquest apartat, s'ha realitzat un agrupament k-means amb 6 grups (K=6) dels llocs web utilitzant un vector els valors mitjos de cada aspecte enregistrat.

L'arxiu *FunctionsForActivityTwoAndThreeDef.py* conté totes les funcions utilitzades per a portar a terme l'agrupament esmentat. Existeixen tres grups de funcions:

1. Funcions d'accés a les dades i càlcul dels valors promig: *readRatingsDictio* i *meanValuations*.

```
def readRatingsDictio(file="outfile.data"):
    lines = f1.readRatings(file)
    dictio = {int(l[0]) : {}  for l in lines}
    for l in lines:
        # l[0] is the web id, l[1] is the user id,
        # l[2..6] are the user's values for each topic.
        # There are an empty field (7) in each register... Is there a mistake?
        valuations = l[2:7]
        dictio[int(l[0])][int(l[1])] = valuations
    return dictio

# Compute the mean of each web to get a list of 5 valuations for each one
def meanValuations(dictio):
    dictioMeans = {}
    floatDictio = {}
    for web in dictio:
        floatDictio[web] = {}
        for user in dictio[web]:
            floatDictio[web][user] = []
            for value in dictio[web][user]:
                floatDictio[web][user].append(float(value))
    for web in floatDictio:
        # Each web has a dictio {user: [5 valuations]}
        vals = zip(*floatDictio[web].values())
        dictioMeans[web] = list(map(lambda x:sum(x)/len(x), vals))
    return dictioMeans
```

Aquestes funcions accedeixen a les dades dels arxius de dades (*newStWebs.data* i *newScWebs.data*) i carreguen el contingut en un diccionari, per a després calcular el vector amb els valors mitjans per a cada aspecte.

2. Funcions auxiliars per a calcular distàncies: *euclideanDist* i *euclideanSimilarity*.

```python
def euclideanDist(list1, list2):
    # Compute the sum of squares of the two lists (should have same length)
    sum2 = sum([pow(list1[i]-list2[i], 2) for i in range(len(list1))])
    return sqrt(sum2)

def euclideanSimilarity(list1, list2):
    return 1/(1+euclideanDist(list1, list2))
```

Calculen la distància Euclidiana entre dos llistats (els de les valoracions mitjanes, en aquest cas). En el cas de la similitud Euclidiana, es calcula la similitud (inversa segura de la distància, per no poder tenir denominador igual a 0) entre dos diccionaris. Ambdues funcions seran utilitzades a la funció *kmeans_list*.

3. Funció k-means: donat un diccionari tipus {idWeb: [valoracions]} retorna un agrupament amb k grups, executant un nombre màxim d'iteracions i segons la funció de similitud especificada. Els resultats es presenten en dos llistats: assignacions i centroides dels grups.

```python
# Given a dictionary like {key1 : [values]} it computes k-means
# clustering, with k groups, executing maxit iterations at most, using
# the specified similarity function.
# It returns two things (as a tuple):
# -{key1:cluster number} with the cluster assignemnts (which cluster
#  does each element belong to
# -[{key2:values}] a list with the k centroids (means of the values
#  for each cluster.
# All values should have the same length, and are interpreted as coordinates
# of the elements to be clustered.
def kmeans_list(dictionary, k, maxit, similarity = euclideanSimilarity):
    # First k random points are taken as initial centroids.
    # Each centroid is [values]
    centroids = [dictionary[x] for x in sample(dictionary.keys(), k)]
    # Assign each key1 to a cluster number
    previous   = {}
    assignment = {}
    # On each iteration it assigns points to the centroids and computes
    # new centroids
    for it in range(maxit):
        # Assign points to the closest centroids
        for key1 in dictionary:
            simils = map(similarity,repeat(dictionary[key1],k), centroids)
            assignment[key1] = simils.index(max(simils))
        # If there are no changes in the assignment then finish
        if previous == assignment:
            break
        previous.update(assignment)
        # Recompute centroids: annotate coords of points in each cluster
        # like {idcluster: [[values1], [values2], ...]
        coords    = {x : [] for x in range(k)}
        for key1 in dictionary:
            group = assignment[key1]
            coords[group].append(dictionary[key1])
        # Compute means (new centroids)
        centroids = []
        for group in coords:
            vals = zip(*coords[group])
            centroids.append(list(map(lambda x:sum(x)/len(x), vals)))
        if None in centroids: break
    return (assignment, centroids)
```

La crida a les funcions es realitza en l'arxiu *ActivityTwoAndThreeDef.py*, que es presenta a continuació. Com es pot veure al codi, l'agrupament s'ha realitzat dues vegades: una amb

l'arxiu de dades escalades amb l'escalat 0-1 i altra amb les dades estandaritzades. Com s'exposarà a l'apartat següent, en ambdós casos, s'ha obtingut un resultat similar.

```
# Activity 2: First Cluster Analysis

import FunctionsForActivityTwo as f2

# Scaled Data Test

ratingsDictio = f2.readRatingsDictio("newScWebs.data")
# print(ratingsDictio)
means = f2.meanValuations(ratingsDictio)
# print(means)
(assignmentSc, centroidsSc) = f2.kmeans_list(means, 6, 20)
print(assignmentSc)

# Standarized Data Test

ratingsDictio = f2.readRatingsDictio("newStWebs.data")
# print(ratingsDictio)
means = f2.meanValuations(ratingsDictio)
# print(means)
(assignmentSt, centroidsSt) = f2.kmeans_list(means, 6, 20)
print(assignmentSt)
```

Activitat 3

En aquest apartat utilitzarem la mesura de qualitat d'agrupament Adjusted Rand Index per a avaluar els dos agrupaments de l'apartat anterior. Aquest índex compara dos agrupaments per veure com de semblants són: pot donar un resultat entre -1 i 1, on un -1 indica que els agrupaments són molt diferents i un 1 indica que són idéntics.

Per a llençar l'Adjusted Rand Index utilitzarem una funció de la llibreria sklearn, tenint en compte que l'agrupament real de les dades és [1..5], [6..11], [12..16], [17..20], [21..24] i [25..30].

La definició del llistat labels_true (referència correcta) i la crida a sklearn.adjusted_rand_score està inclosa a l'arxiu *ActivityTwoAndThreeDef.py*, i es presenta a continuació:

```
# Activity 3: Adjusted Rand Index
from sklearn import metrics

# Reference clustering
labels_true = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5]

# Scaled
print(metrics.adjusted_rand_score(labels_true, list(assignmentSc.values())))

# Standarized
print(metrics.adjusted_rand_score(labels_true, list(assignmentSt.values())))
```

Els resultats de l'execució han sigut:

- Per a les dades escalades: 0,08
- Per a les dades estandaritzades: 0,12

Ambdós resultats són similars, atès que la distribució de les dades originals era propera a una distribució normal (i per tant també ho era la distribució de les dades escalades). En qualsevol cas, el resultat sembla llunyà del llistat de referència, el que pot fer dubtar de la qualitat de l'agrupament.

Els resultats en execucions successives han presentat una dispersió a l'Adjusted Rand Index de fins a ±0,1, i sempre s'ha mantingut proper utilitzant ambdos mètodes.

Activitat 4

En aquest apartat construirem un recomanador que realitzi una ponderació de l'activitat dels usuaris a cada web, tenint en compte la similitud d'aquest amb l'usuari objecte de la recomanació. El resultat de l'execució del programa serà un llistat que, per a cada usuari, recollirà una ordenació de les webs segons l'afinitat estimada. Per tal d'avaluar la qualitat de la recomanació, trobarem la posició mitjana a la qual es troba el favorit de l'usuari, que ve llistat a l'arxiu original *favorits.data*.

Les funcions necessàries per a executar el recomanador es recullen a l'arxiu *FunctionsForActivityFourDef.py*, i s'agrupen en 3 grups:

- Funcions de lectura de dades: readUserValuations i readFavorites. Llegeixen els fitxers de dades *(newScWebs.data* i *newStWebs.data*) i de favorits (*favorits.data*), respectivament.

```
# Reads webs.data and returns a dictionary of each web user's values as:
# {user: {web:[list of values]}}
def readUserValuations(filename="webs.data"):
    myfile = file(filename)
    lines = [(l.strip()).split("\t") for l in myfile.readlines()]
    # l[1] is the user id
    dictio = {int(l[1]) : {}  for l in lines}
    for l in lines:
        # l[0] is the web id, l[2..6] are the user's values.
        valuations = []
        for x in l[2:]:
            valuations.append(float(x))
        dictio[int(l[1])][int(l[0])] = valuations
    return dictio

# Read favorits.data file, return {user: favorite web}
def readFavorites(filename="favorits.data"):
    myfile = file(filename)
    lines = [(l.strip()).split("\t") for l in myfile.readlines()]
    # l[0] is the user id, l[1] the web id
    dictio = {int(l[0]) : int(l[1])  for l in lines}
    return dictio
```

- Funcions de càlcul de similitud (coeficient de Pearson): calculen el coeficient de Pearson entre dos llistats, entre dos usuaris d'un diccionari o entre tots els elements d'un diccionari, respectivament.

```
# A simple Pearson correlation function between two lists
def simplePearson(list1, list2):
    mean1 = sum(list1)
    mean2 = sum(list2)
    num  = sum([(list1[i]-mean1)*(list2[i]*mean2) for i in range(len(list1))])
    den1 = sqrt(sum([pow(list1[i]-mean1, 2) for i in range(len(list1))]))
    den2 = sqrt(sum([pow(list2[i]-mean2, 2) for i in range(len(list2))]))
    den  = den1*den2
    if den==0:
        return 0
    return num/den
```

```
# Compute the mean Pearson coeff between a pair of users
# Input two diccionaries with ratings of each user {web: [valuations]}
def pearsonCoeff_list(user1, user2):
    # Retrieve the webs common to both users
    commons  = [x for x in user1 if x in user2]
    nCommons = float(len(commons))
    # If there are no common elements, return zero; otherwise
    # compute the coefficient
    if nCommons==0:
        return 0
    # Compute the mean Pearson coeff for all common webs
    return sum([simplePearson(user1[x], user2[x]) for x in commons])/nCommons

def pearsonCoeff(dic1, dic2):
    # Retrieve the elements common to both dictionaries
    commons  = [x for x in dic1 if x in dic2]
    nCommons = float(len(commons))

    # If there are no common elements, return zero; otherwise
    # compute the coefficient
    if nCommons==0:
        return 0

    # Compute the means of each dictionary
    mean1 = sum([dic1[x] for x in commons])/nCommons
    mean2 = sum([dic2[x] for x in commons])/nCommons

    # Compute numerator and denominator
    num  = sum([(dic1[x]-mean1)*(dic2[x]-mean2) for x in commons])
    den1 = sqrt(sum([pow(dic1[x]-mean1, 2) for x in commons]))
    den2 = sqrt(sum([pow(dic2[x]-mean2, 2) for x in commons]))
    den  = den1*den2

    # Compute the coefficient if possible or return zero
    if den==0:
        return 0

    return num/den
```

- Recomanador: calcula la similitud entre usuaris, per a després construir un diccionari que, per a cada web, calculi la valoració promig ponderada pel pes d'aquesta similitud, en el format {webId: [valor*similitud]} / {webId: [similitud]}. En l'últim pas, ordena els resultats obtinguts en funció d'aquesta valoració ponderada.

```
# Produces a sorted list of weighted ratings from a dictionary of
# user ratings and a user id.
# You can choose the function of similarity between users.
def weightedRating(dictio, user, similarity = pearsonCoeff):
    # In the first place a dictionary is generated with the similarities
    # of our user with all other users.
    # This dictionary could be stored to avoid recomputing it.
    simils = {x: similarity(dictio[user], dictio[x])
              for x in dictio if x != user}

    # Auxiliary dictionaries {webId: [rating*users similarity]}
    # and {webId: [users similarity]} (numerator and denominator
    # of the weighted rating)
    numerator   = {}
    denominator = {}

    # The ratings dictionary is traversed, while filling the auxiliary
    # dictionaries with the values found.
    for userId in simils:
        for webId in dictio[userId]:
            if not numerator.has_key(webId):
                numerator  [webId] = []
                denominator[webId] = []
            s = simils[userId]
            numerator  [webId].append(sum(dictio[userId][webId])*s)
            denominator[webId].append(s)
```

```
    # Compute and sort weighted ratings
    result = []
    for webId in numerator:
        s1 = sum(numerator  [webId])
        s2 = sum(denominator[webId])
        if s2 == 0:
            mean = 0.0
        else:
            mean = s1/s2

    # Append the rating only if the user does not have it already
    if not dictio[user].has_key(webId):
            result.append((webId,mean))

    result.sort(key = lambda x: x[1], reverse=True)
    return result
```

Totes les funcions es criden des de l'arxiu *ActivityFourDef.py*, recollit a continuació. En aquest apartat s'han utilitzat les dades escalades (arxiu *newScWebs*.data).

```
import FunctionsForActivityFourDef as f4

# We will use the scaled values for this activity
valuations = readUserValuations("newScWebs.data")
favorites = readFavorites("favorits.data")

# Run the recommender. We get {user: (list of recommended webs in desc order)}
recomPearson = {usr : zip(*weightedRating(valuations, usr, pearsonCoeff_list))[0] for usr in valuations.keys() }

# print(recomPearson)

# Finally compute the average position of the favorite web in the
# recommendation for each user. The closest to position 0, the better the
# recommender has performed
positions = [recomPearson[usr].index(favorites[usr]) for usr in favorites]

meanPosition = sum(positions)/float(len(positions))

print(meanPosition)
```

La posició promig calculada ha sigut 6,62 (7,62, tenint en compte que el llistat comença a 0). De nou, s'ha realitzat la prova amb les dades estandaritzades (arxiu *newStWebs*.data), obtenint un pitjor resultat (8,60).

En ambdós casos el resultat fa dubtar de la qualitat del recomanador utilitzat. Cabria la millora modificant la funció de distància o realitzant algun ajust de les dades distint de la normalització, que tingui en compte la propensió dels usuaris (usuaris que siguin més actius en general i que, per tant, puguin tenir inflades les seves activacions a totes les webs).

Conclusió i comentaris

La valoració global de l'aprenentatge en aquesta PAC és positiva, tot i que els resultats no siguin tan ajustat com, a priori, m'esperava. Esperaré la resolució correcta per a treure conclusions sobre la qualitat del recomanador desenvolupat. Ha sigut de gran ajuda tenir la resolució de la pràctica de l'any anterior com a referència, per sortir de situacions de bloqueig que d'altra manera s'haguessin traduït en consultes al fòrum de l'assignatura o a l'email dels consultors, en tot cas amb una resposta no immediata. Per últim, mencionar que vaig tenir problemes de compatibilitat al executar algunes parts del codi en Python 3.4, sobretot relacionades amb el comportament de la funció map i amb la crida a mòduls de funcions. Per aquest motiu, vaig decidir executar tot el contingut de la pràctica en un entorn virtual de Python 2.7.