



***BENEMÉRITA UNIVERSIDAD
AUTÓNOMA DE PUEBLA***

***FACULTAD DE CIENCIAS DE LA
COMPUTACIÓN***

PROFESOR: ARTURO OLVERA LÓPEZ

***ALUMNO: ALAN JAMIT MARCIAL
MARÍN***

***MATERIA: MINERÍA DE DATOS
PROYECTO***

INTRODUCCIÓN:

Retomando el documento descriptivo, el presente proyecto tratará, sobre un ensamble básico de clasificadores el cuál estará relacionado con: Las fases de Pre-procesamiento, extracción de conocimiento y visualización dentro del proceso KDD.

Para su puesta en marcha, debe hacerse considerando la técnica PCA, previamente vista en clases, para la reducción de dimensionalidad y la extracción de conocimiento.

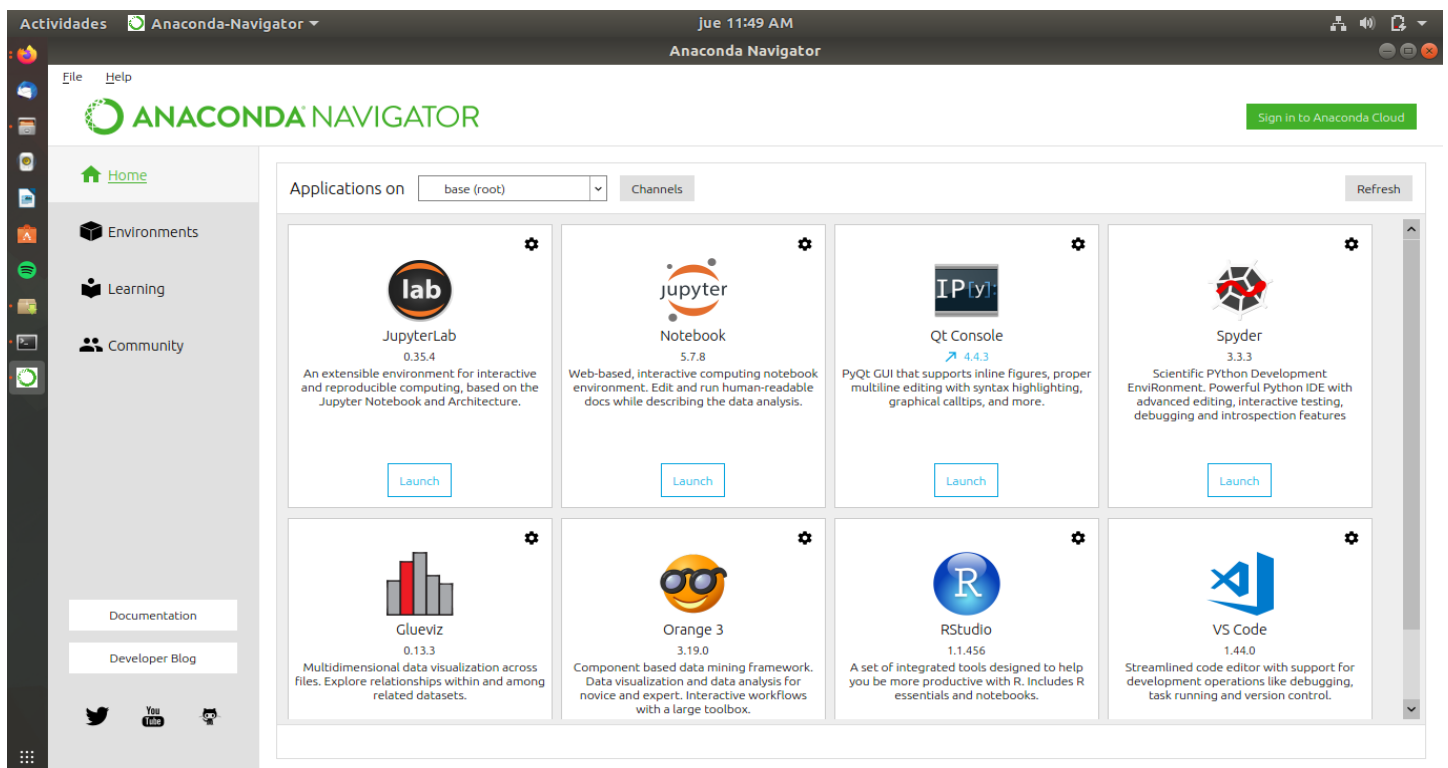
Ahora bien manos a la obra.

-ELECCIÓN DE UNA IDE:

El primerísimo paso fué, antes que nada, la elección de la IDE con la que quería trabajar, tomando en cuenta los parámetros indicados que debía cumplir el proyecto, su facilidad de uso, que tan completa es y finalmente si me serviría para hacer todo lo necesario.

Programas de este tipo hay bastantes, como por ejemplo: Weka u Orange, recomendaciones del profesor, además de haber sido previamente usadas en clase.

Después de haberlas curioseado un tiempo, hacer las conversiones correspondientes y revisar manuales de ambas IDE's me di cuenta que no era capaz de adaptarme a la interfaz de alguna de ellas, por lo que decidí seguir la recomendación personal de un amigo, la cuál fue usar Anaconda, en este caso la versión para Linux.



En estas primeras impresiones me di cuenta que Anaconda es una IDE que a su vez engloba a otras varias IDE's dentro suyo, entre ellas la propia Orange, así como Jupyter Lab, Jupyter Notebook, Spyder, entre otras. Es aquí donde me decanto por usar Jupyter Notebook. ¿Por qué Jupyter Notebook?, porque es una herramienta muy parecida a otras que ya he utilizado (XAMPP, phpMyAdmin, Git Hub, ETC), tanto en su funcionalidad, como en su interfaz, por lo cuál estoy ampliamente familiarizado.

aplicación del algoritmo "AdaBoost" (el cual es considerado por algunos como el algoritmo de aprendizaje mejor supervisado.), además de darnos a entender en que consiste el "Bagging" y "Boosting" respectivamente, explicaré brevemente cada uno.

-Bagging

El método de agregación de Bootstrap, mejor conocido como Bagging, es una técnica en la que los datos son tomados del conjunto de datos original "S" veces para hacer "S" nuevos conjuntos de datos.

Los conjuntos de datos son del mismo tamaño del original, cada conjunto de datos se crea seleccionando de forma aleatoria un ejemplo del original con reemplazo.

Se quiere dar a entender que "con un reemplazo" se puede seleccionar el mismo ejemplo más de una vez, dicha propiedad permite tener valores en el nuevo conjunto de datos que se repiten y algunos valores del original no estarán presentes en el nuevo set.

Después de construir los conjuntos de datos "S", se aplica un algoritmo de aprendizaje a cada individuo, cuando se desee clasificar una nueva pieza de datos, se aplicarán los clasificadores "S" a la nueva pieza de datos y así tener un voto mayoritario.

-Boosting

Es una técnica similar al Bagging, una de las características que comparten es que ambas usan siempre el mismo tipo de clasificador, pero en el Boosting los diferentes clasificadores son entrenados secuencialmente. Cada nuevo clasificador es entrenado basándose en el rendimiento de los que ya están entrenando. El Boosting hace que los nuevos clasificadores se centren en los datos que se clasificaron de manera errónea anteriormente.

Aún así el Boosting es diferente del Bagging debido a que la salida se calcula a partir de una suma ponderada de todos los clasificadores, los pesos no son iguales a los del Bagging, sino que se basan en que tan exitoso fue el clasificador en la iteración anterior.

Existen muchas versiones del Boosting pero el más popular es la versión llamada AdaBoost.

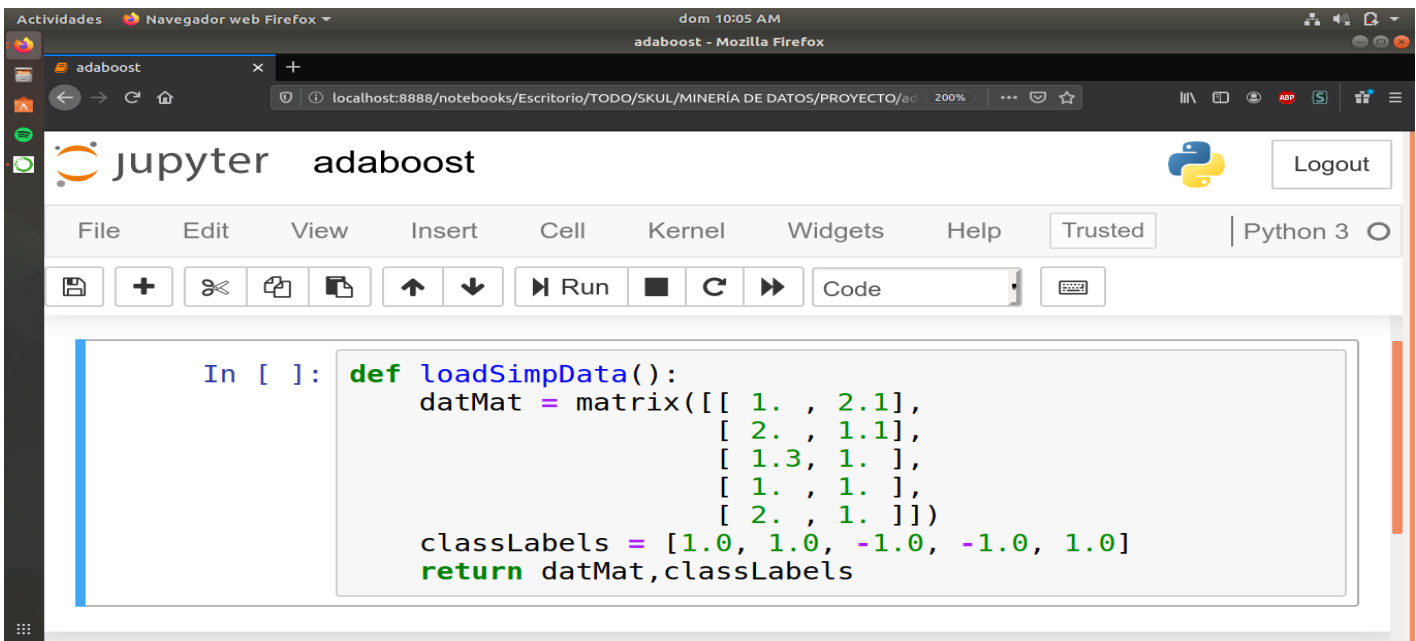
Aquí un ejemplo como muchos otros de la creación y puesta en marcha del algoritmo AdaBoost, este ejemplo fue el que me pareció más interesante y del que decidí investigar y documentar un poco más.

1.- Creando un "learner" con un "decision stump".

Un "decision stump" es un árbol de decisión simple, sabiendo como funcionan éstos últimos, ahora haremos que un decision stump tome una decisión sobre una sola función, será un árbol con una sola decisión por lo que a este se le llamará solamente "stump" o por su traducción tocón.

Mientras se construye el algoritmo AdaBoost solamente se trabajará sobre un conjunto de datos simple, para asegurarnos que todo está en orden.

Ahora se creará un archivo llamado adaboost.py

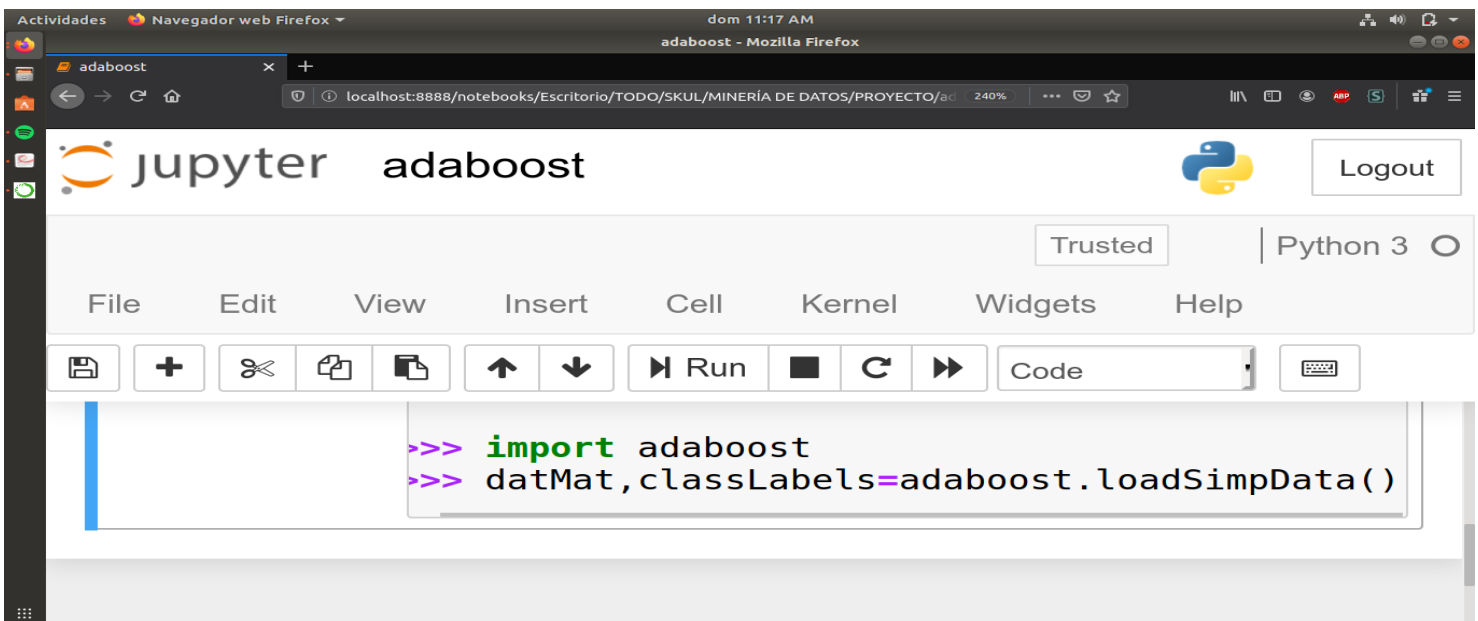


```
In [ ]: def loadSimpData():
        datMat = matrix([[ 1. , 2.1],
                           [ 2. , 1.1],
                           [ 1.3, 1. ],
                           [ 1. , 1. ],
                           [ 2. , 1. ]])
        classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]
        return datMat, classLabels
```

Ahora intentaremos elegir un valor en un eje que separe totalmente los círculos de los cuadrados, aunque esto de hecho no es posible, sino que se trata del famoso problema "45" en el que los árboles de decisión son bien conocidos por tener dificultades.

Así pues Adaboost necesitará prescindir de varios stumps para clasificar adecuadamente el conjunto de datos. Mediante el uso de estos stumps, se puede construir un clasificador para clasificar completamente los datos.

Podremos cargar el conjunto de datos y las etiquetas escribiendo la siguiente línea de código:



```
>>> import adaboost
>>> datMat, classLabels = adaboost.loadSimpData()
```

Ahora que tenemos el conjunto de datos cargado, podemos crear algunas funciones para la construcción de nuestro stump.

```
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = ones((shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray

def buildStump(dataArr,classLabels,D):
    dataMatrix = mat(dataArr); labelMat = mat(classLabels).T
    m,n = shape(dataMatrix)
    numSteps = 10.0; bestStump = {}; bestClasEst = mat(zeros((m,1)))
    minError = inf
    for i in range(n):
        rangeMin = dataMatrix[:,i].min(); rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps
        for j in range(-1,int(numSteps)+1):
            for inequal in ['lt', 'gt']:
                threshVal = (rangeMin + float(j) * stepSize)
                predictedVals = \
                    stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr = mat(ones((m,1)))
                errArr[predictedVals == labelMat] = 0
                weightedError = D.T*errArr
                #print "split: dim %d, thresh %.2f, thresh inequal: \
                %s, the weighted error is %.3f" %\
                    (i, threshVal, inequal, weightedError)
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst
```

El siguiente código contiene dos funciones, la primera función "stumpClassify()", realiza una comparación de umbral para clasificar datos. Todo en el lado del "thresh-old" se arrolará a la clase "-1", y todo lo que quede en el lado restante se arrojará en la clase "+1".

Esta se realiza mediante el filtrado de matriz, primero configurando la matriz de retorno a todos los "1's" y luego configurando valores que no cumplen con la desigualdad a -1, podemos hacer esta comparación en cualquier función dentro del conjunto de datos y además se puede cambiar de > a <.

La siguiente función "buildStump()", iterará sobre todas las entradas posibles para la función "stumpClassify()" y encontrará el mejor stump para el conjunto de datos. Luego la mejor opción será con respecto al vector de peso de datos D.

La función comienza con asegurarnos de que los datos de entrada estén en el formato correcto para las matrices, después se crea un diccionario vacío con el nombre "bestStump" el cuál será usado para almacenar la información del clasificador correspondiente a la mejor opción tomada por el stump, dado el vector de peso D. La variable "numSteps" se usará para iterar sobre los posibles valores de las características, también inicializa la variable "minError" a un infinito positivo, esta variable se usa para encontrar el más mínimo error posible más adelante.

La parte principal del código está triplemente anidada para bucles, la primera parte pasa por encima de todas las características de nuestro conjunto de datos, debemos considerar valores numéricos y calcular el máximo y mínimo para ver qué tan grande debe ser su tamaño.

Dentro de los tres bucles anidados, se llama a la función "stumpClassify()" con el conjunto de datos y sus tres variables de bucle, así "stumpClassify()" devuelve su predicción de clase basada en estas variables de bucle, a continuación, crea el vector de columna "errArr", que contiene un 1 para cualquier valor en "predictedVals" que

no es igual a la clase actual en “labelMat”, multiplicamos estos errores por los pesos **D** y se suman los resultados para darle un sólo número: el “weightedError”.

Es aquí donde AdaBoost interactúa con el clasificador, evaluamos nuestro clasificador basado en los pesos **D** y no en otra medida de error, si deseamos utilizar otro clasificador, debemos incluir este cálculo para definir el mejor clasificador para **D**. Ahora imprimiremos todos los valores y como último se compara el error con su error mínimo conocido y si está debajo de éste guardaremos este stump en el diccionario “bestStump”. Tanto el diccionario, como el error y las estimaciones de la clase, van todas de vuelta al algoritmo AdaBoost.

PROBLEMA A RESOLVER:

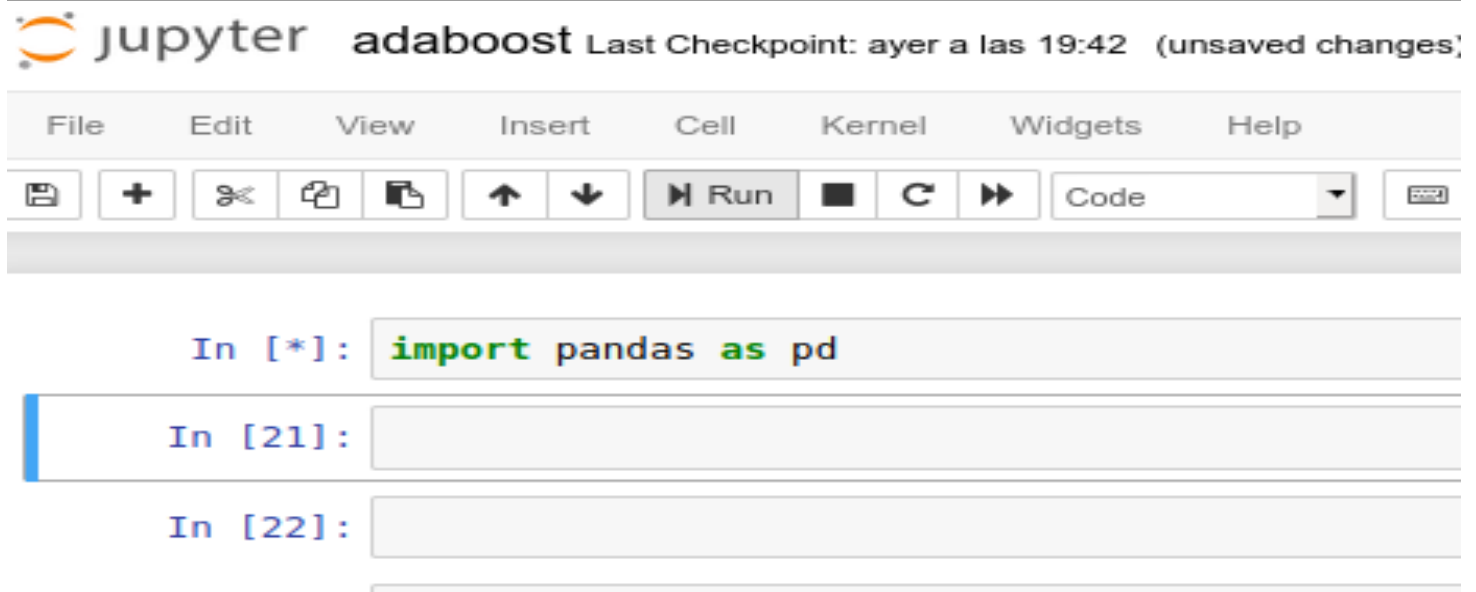
Después de haber visto varios ejemplos del algoritmo AdaBoost podemos usar lo aprendido para poner en marcha nuestro proyecto.

Dicho proyecto nos pide implementar un ensamble básico de clasificadores, nuestro programa deberá aceptar como entrada un archivo de texto simple, el cual tiene 420 elementos, cada uno descrito por 19 atributos y se tienen 7 clases distintas con valores del 0 al 6.

Ahora empezaremos a manipular dicho archivo de texto para ir implementando los cambios necesarios para comenzar con el ensamble de clasificadores.

PRIMER PASO:

Importamos la biblioteca “pandas” y la declaramos como “pd”.



Ahora declaramos la función “pdf_text” para extraer el texto del documento “data.txt” que alberga los valores del ejemplo de conjunto de datos.

```
In [12]: import pandas as pd
```

```
In [13]: pdf_txt = 'data.txt'
```


Una vez hecho esto, el siguiente paso será usar la función “openfile” para realizar la obtención del texto contenido en el archivo “data.txt”.

```
In [12]: import pandas as pd
```

```
In [13]: pdf_txt = 'data.txt'
```

```
In [14]: openfile = open(pdf_txt, 'r')
```

Imprimimos los datos de nuestro archivo:

jupyter adaboost Last Checkpoint: hace unos segundos (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [6]: for line in openfile:
        print (line)
```

```
180,141,0,0,0,0.055555563,0.13608278,0.055555556,0.13608277,0.037037037,0,0.11111111,0,-0.11111111,0.22222222,-0.11111111,0.11111111,0.11111111,-2.0943952,6

225,58,0,0,0,0.3333335,0.42163706,0.4444445,0.34426522,8.333333,5.5555553,14.111111,5.3333335,-8.333333,17.333334,-9,14.111111,0.62222224,-2.0685637,6


170,154,0,0,0,0.11111111,0.17213261,0.055555556,0.13608277,0.074074075,0,0.22222222,0,-0.22222222,0.44444445,-0.22222222,0.22222222,0.22222222,-2.0943952,6

238,61,0,0,0,0.66666657,0.4714045,0.7222221,0.49065334,8.777778,5.888889,14.333333,6.111111,-8.666667,16.666666,-8,14.333333,0.5888227,-2.1218371,6

189,62,0,0,0,0.38888893,0.2509241,0.44444442,0.3442651,7.6296296,5,13.111111,4.7777777,-7.888889,16.444445,-8.55555,13.111111,0.6359381,-2.0669532,6

242,57,0,0,0,0.11111111,2.444444,2.3538773,9.277778,2.7601657,15.37037,11.333333,20.88889,13.888889,-12.111111,16.55555,-4.4444447,20.88889,0.46630767,-2.3706133,6
```

Volvemos a imprimir nuestros datos pero esta vez ordenados en una tabla, utilizando la función “head()” para obtener las primeras 5 posiciones de la tabla.

jupyter adaboost Last Checkpoint: hace 11 minutos (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [16]: df = pd.read_csv("data.txt", sep = ',', header = None, thousands = ',')
        df.head()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0	140	125	0	0.0	0.0	0.277778	0.062963	0.666667	0.311111	6.185185	7.333334	7.666666	3.555556	3.444444	4.444445	-7.888889	7.777778	0.545635
1	188	133	0	0.0	0.0	0.333333	0.266667	0.500000	0.077778	6.666666	8.333334	7.777778	3.888889	5.000000	3.333333	-8.333333	8.444445	0.538580
2	105	139	0	0.0	0.0	0.277778	0.107407	0.833333	0.522222	6.111111	7.555555	7.222222	3.555556	4.333334	3.333333	-7.666666	7.555555	0.532628
3	34	137	0	0.0	0.0	0.500000	0.166667	1.111111	0.474074	5.851852	7.777778	6.444445	3.333333	5.777778	1.777778	-7.555555	7.777778	0.573633
4	39	111	0	0.0	0.0	0.722222	0.374074	0.888889	0.429629	6.037037	7.000000	7.666666	3.444444	2.888889	4.888889	-7.777778	7.888889	0.562919

Además podemos ver que las columnas de nuestra tabla se dividen en 19, que son los 19 atributos que describen a cada elemento del conjunto, lo cual hemos estipulado previamente.

jupyter adaboost Last Checkpoint: hace 15 minutos (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [16]: `df = pd.read_csv("data.txt", sep = ',', header = None, thousands = ',')`
`df.head()`

Out[16]:

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0.0	0.0	0.277778	0.062963	0.666667	0.311111	6.185185	7.333334	7.666666	3.555556	3.444444	4.444445	-7.888889	7.777778	0.545635	-1.121818	0	
0	0.0	0.0	0.333333	0.266667	0.500000	0.077778	6.666666	8.333334	7.777778	3.888889	5.000000	3.333333	-8.333333	8.444445	0.538580	-0.924817	0	
0	0.0	0.0	0.277778	0.107407	0.833333	0.522222	6.111111	7.555555	7.222222	3.555556	4.333334	3.333333	-7.666666	7.555555	0.532628	-0.965946	0	
0	0.0	0.0	0.500000	0.166667	1.111111	0.474074	5.851852	7.777778	6.444445	3.333333	5.777778	1.777778	-7.555555	7.777778	0.573633	-0.744272	0	
0	0.0	0.0	0.722222	0.374074	0.888889	0.429629	6.037037	7.000000	7.666666	3.444444	2.888889	4.888889	-7.777778	7.888889	0.562919	-1.175773	0	

Posterior a esto, utilizamos la función "df.tail()" para visualizar los últimos 5 elementos de la tabla.

Tomando en cuenta que nuestro programa empieza a contar desde el 0, podemos ver que en efecto la tabla posee 420 elementos, tal y como está especificado anteriormente.

jupyter adaboost Last Checkpoint: hace 29 minutos (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [17]: `df.tail()`

Out[17]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
415	225	58	0	0.0	0.000000	0.333334	0.421637	0.444444	0.344265	8.333333	5.555555	14.111111	5.333334	-8.333333	17.333334	-9.000000	14.111
416	170	154	0	0.0	0.000000	0.111111	0.172133	0.055556	0.136083	0.074074	0.000000	0.222222	0.000000	-0.222222	0.444444	-0.222222	0.222
417	238	61	0	0.0	0.000000	0.666667	0.471405	0.722222	0.490653	8.777778	5.888889	14.333333	6.111111	-8.666667	16.666666	-8.000000	14.333
418	189	62	0	0.0	0.000000	0.388889	0.250924	0.444444	0.344265	7.629630	5.000000	13.111111	4.777778	-7.888889	16.444445	-8.555555	13.111
419	242	57	0	0.0	0.111111	2.444444	2.353877	9.277778	2.760166	15.370370	11.333333	20.888890	13.888889	-12.111111	16.555555	-4.444445	20.888

Después de esto podemos usar la función "df.dtypes()" para ver los tipos de atributos que existen dentro del conjunto de datos, en este caso son solo dos tipos: int64 y float64, en los que se dividen los 19 atributos.

In [18]: df.dtypes

```
Out[18]: 0      int64
1      int64
2      int64
3     float64
4     float64
5     float64
6     float64
7     float64
8     float64
9     float64
10    float64
11    float64
12    float64
13    float64
14    float64
15    float64
16    float64
17    float64
18    float64
19     int64
dtype: object
```

Defino un arreglo llamado "cabecera" el cual contiene el nombre y número de cada columna, esto lo hice para identificarlas dentro de la tabla.

```
In [19]: cabecera = ["column1", "column2", "column3", "column4", "column5", "column6", "column7", "column8", "column9", "column10", "column11", "column12", "column13", "column14", "column15", "column16"]
df.columns = cabecera
df.head()
```

```
Out[19]:
```

	column1	column2	column3	column4	column5	column6	column7	column8	column9	column10	column11	column12	column13	column14	column15	column16
0	140	125	0	0.0	0.0	0.277778	0.062963	0.666667	0.311111	6.185185	7.333334	7.666666	3.555556	3.444444	4.444445	-7.888889
1	188	133	0	0.0	0.0	0.333333	0.266667	0.500000	0.077778	6.666666	8.333334	7.777778	3.888889	5.000000	3.333333	-8.333333
2	105	139	0	0.0	0.0	0.277778	0.107407	0.833333	0.522222	6.111111	7.555555	7.222222	3.555556	4.333334	3.333333	-7.666666
3	34	137	0	0.0	0.0	0.500000	0.166667	1.111111	0.474074	5.851852	7.777778	6.444445	3.333333	5.777778	1.777778	-7.555555
4	39	111	0	0.0	0.0	0.722222	0.374074	0.888889	0.429629	6.037037	7.000000	7.666666	3.444444	2.888889	4.888889	-7.777778

Ahora uso "df.describe()" para hacer la descripción de las propiedades que posee el conjunto, tales como por ejemplo: valores máximos y valores mínimos por columna, la varianza, la media, entre otras cosas.

In [20]: df.describe()

Out[20]:

	column1	column2	column3	column4	column5	column6	column7	column8	column9	column10	column11	column12	col
count	420.000000	420.000000	420.0	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000	420.000000
mean	144.245238	142.345238	0.0	0.011905	0.005820	1.756878	3.465178	2.357937	6.538579	30.833069	27.335979	36.672752	28.465238
std	73.219449	60.798955	0.0	0.035253	0.029154	2.381124	30.810966	3.651203	68.978325	31.442611	28.866983	36.478432	29.465238
min	1.000000	11.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	85.000000	92.500000	0.0	0.000000	0.000000	0.666668	0.414811	0.722222	0.429630	7.027778	6.111111	8.638889	4.638889
50%	160.000000	152.000000	0.0	0.000000	0.000000	1.333334	0.923948	1.555556	1.068055	18.018518	15.222222	21.277777	19.222222
75%	205.000000	190.000000	0.0	0.000000	0.000000	2.166667	1.705687	2.833333	1.968498	48.342592	43.805557	59.222220	41.922222
max	253.000000	251.000000	0.0	0.222222	0.222222	25.500000	572.996400	44.722225	1386.329200	143.444440	136.888890	150.888890	142.523810

##Hasta aquí logré llevar a cabo mi avance, sé que no es mucho, pero debido a la situación y mi poco o nulo conocimiento y/o experiencia en este tipo de algoritmos, entre otros temas de la asignatura, trato de hacer mi mejor esfuerzo, teniendo en cuenta también las entregas que debo hacer en las demás asignaturas, espero su comprensión y estaré pendiente para la fecha del siguiente avance.##