Fundamentos de Aprendizaje Automático 2019/2020

PRÁCTICA DE INTRODUCCIÓN

1. Objetivos

El objetivo de esta práctica es doble, por un lado se busca familiarizarnos con el lenguaje de programación Python y, por otro, se deberá realizar parte de la clase *Datos*, una de las que se empleará durante el curso para gestionar los diferentes datasets con los que probar los algoritmos de clasificación a estudiar.

2. Preliminares

Python es uno de los lenguajes que está adquiriendo mayor popularidad en el campo del reconocimiento de patrones[1] y del aprendizaje automático en general. Dada su versatilidad y sencillez, es el lenguaje que vamos a utilizar a lo largo de las prácticas. Junto con el lenguaje propiamente dicho emplearemos también los Jupyter Notebooks para presentar los resultados. Puede desarrollarse el código con el método que se considere más adecuado, tanto de forma autónoma como mediante distribuciones que incluyan IDEs o cualquier otra facilidad. Una opción en este caso es Anaconda, una distribución gratuita de Python que incluye más de 300 paquetes como NumPy, para computación científica básica y análisis de datos, y Scikit-learn, paquete con diversos algoritmos de aprendizaje automático. Además, Anaconda incluye también los mencionados Jupyter Notebooks, que permiten combinar celdas de texto y código, y constituyen el método más apropiado para mostrar de forma conjunta las implementaciones de código realizadas y la discusión de los resultados obtenidos.

Anaconda se puede descargar desde https://www.continuum.io/. **En estas prácticas utilizaremos la versión 3.7**.

IMPORTANTE. Para poder usar Anaconda en los laboratorios de prácticas, debe establecerse la variable de entorno PATH como:

```
# export PATH=${PATH}:/opt/<ruta-distribucion-anaconda>/bin
```

Donde `<ruta-distribucion-anaconda>` dependerá de la versión que estemos instalando. Para arrancar el intérprete de IPython de Anaconda, se debe ejecutar el binario que se encuentra en el directorio de instalación de Anaconda (`/opt/<ruta-distribucion-anaconda>/`):

```
# /opt/<ruta-distribucion-anaconda>/bin/ipython
```

Y para arrancar el servidor de Notebooks, este se debe ejecutar desde el directorio de instalación de la siguiente forma:

```
# cd /opt/<ruta-distribucion-anaconda>
# ./bin/ipython notebook $HOME
```

Las diapositivas disponibles en *Moodle* proporcionan una introducción a Python (variables y tipos de datos, funciones, secuencias de control, ficheros, clases, módulos, paquete NumPy, …).

[1] http://machinelearningmastery.com/best-programming-language-for-machine-learning/

Se recomienda leer esta introducción con detalle y hacer los ejercicios propuestos para adquirir los conocimientos básicos necesarios para implementar la primera de las clases pedidas. Son especialmente importantes los ejercicios que se proponen con el paquete NumPy para comprender las particularidades del trabajo con matrices en estas prácticas.

3. Clase Datos

Los algoritmos de aprendizaje automático permiten construir modelos a partir de un conjunto de datos (datasets). En las prácticas utilizaremos algunos datasets del repositorio de la Universidad de California Irvine (UCI)[2]. Los datasets constarán de un fichero de datos (extensión `.data`) y, generalmente, un fichero con una descripción de los mismos (extensión `.names`). En caso de ser necesario, cada fichero de datos debe tratarse para que siga la siguiente estructura:

- 1ª Fila: Número de datos del conjunto
- 2ª Fila: Nombres de los atributos. El último atributo corresponderá a la clase en el caso de problemas de aprendizaje supervisado.
- 3ª Fila: Tipos de los atributos: Nominal o Continuo
- Resto de filas: Conjunto de datos, uno por fila y campos separados por comas.

Como ejemplo, en esta práctica se proporcionan en *Moodle* los ficheros correspondientes a los conjuntos de datos ***tic-tac-toe*** (http://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame) y ***statlog*** (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29).

En esta práctica nos ocuparemos de implementar el tratamiento preliminar de estos dataset mediante la clase *Datos*, que leerá los datos del fichero de entrada y almacenará la información necesaria para su posterior uso por los algoritmos de aprendizaje automático y métodos de particionado. Una posible estructura de implementación se comenta en el siguiente apartado.

3.1 Diseño

Con el objetivo de desarrollar una aplicación lo más flexible y general posible se plantea la siguiente estructura para la clase *Datos*:

```
import numpy as np

class Datos:

  TiposDeAtributos=('Continuo','Nominal')

  # TODO: procesar el fichero para asignar correctamente las variables
tipoAtributos, nombreAtributos, nominalAtributos, datos y diccionarios
  # NOTA: No confundir TiposDeAtributos con tipoAtributos
  def __init__(self, nombreFichero):

  # TODO: implementar en la practica 1
  def extraeDatos(self, idx):
    pass
```

[2] http://archive.ics.uci.edu/ml/

En esta práctica de introducción se deberá implementar únicamente el constructor de la clase *(__init__)* que recibe como parámetro el nombre del fichero de datos. La función *extraeDatos* se implementará en la práctica 1, por lo que no hay que desarrollarla ahora (la sentencia *pass* permite diferir la implementación).

Los atributos de la clase *Datos* deberán guardar la siguiente información:

- **`TiposDeAtributos`**: esta variable de clase no debe modificarse. Guarda las dos posibles descripciones de los tipos de atributos con los que vamos a trabajar.
- **`tipoAtributos`**: Lista con la misma longitud que el número de atributos del problema (incluyendo la clase) y que contendrá el tipo de atributo de cada variable (Continuo o Nominal). En caso de que el fichero de datos contenga algún tipo de datos que no se corresponda a uno de estos dos tipos, se deberá informar del error. Por ejemplo, se puede lanzar una excepción del tipo `ValueError`.
- **`nombreAtributos`**: Lista con la misma longitud que el número de atributos del problema (incluyendo la clase) y que contendrá el nombre de cada variable. No tiene mayor trascendencia que su posible uso a la hora de presentar resultados.
- **`nominalAtributos`**: Lista de valores booleanos con la misma longitud que el número de atributos del problema (incluyendo la clase) que contendrá *True* en caso de que el atributo sea nominal y False en caso contrario. Esta estructura será útil posteriormente para el uso del paquete Scikit-learn.
- **`datos`**: Array bidimensional de NumPy (matriz) que se utilizará para almacenar los datos. El uso de NumPy facilitará el indexado de los datos, así como el uso del paquete Scikit-learn.
- **`diccionarios`**: lista de diccionarios con la misma longitud que el número de atributos. La posición i-ésima de la lista contendrá el diccionario asociado al atributo i-ésimo. Los diccionarios solo deben generarse para las variables nominales, ya que para las variables continuas se guardará un diccionario vacío. En el caso de las variables nominales, el diccionario establecerá la relación entre los valores nominales o categóricos (claves) y un entero (valor). Para garantizar que este mapeo es consistente para diferentes ficheros y permutaciones de los datos, los enteros deben asignarse en orden lexicográfico de las claves. Así, por ejemplo, en el caso del conjunto de datos tic-tac-toe, donde el primer atributo puede tomar los valores 'x', 'b', 'o', el diccionario correspondiente deberá ser: `{'b': 0, 'o': 1, 'x': 2}`. Una vez obtenidos los diccionarios de cada atributo, las variables categóricas deberán ser codificadas a los valores enteros asignados en el diccionario para su posterior almacenamiento en el array NumPy `datos`. Como veremos a lo largo de las prácticas, el uso de estas matrices, con valores numéricos, facilita mucho la programación y acelera la ejecución de los algoritmos. En este sentido, la combinación entre la matriz `datos` y el `diccionario` nos permite saber cuáles son los valores categóricos originales asociados a cada valor numérico y usar ese valor numérico para los cálculos matemáticos.

La clase se definirá en el módulo *Datos* (fichero **Datos.py**). De esta forma, se pueden instanciar elementos de la clase como sigue, dentro de un módulo Main (fichero **MainDatos.py**) que prueba esta clase inicial y cualquiera de las posteriores.

```
from Datos import Datos
dataset=Datos('<datasets-home>/tic-tac-toe.data')
```

Tanto la plantilla de la clase Datos, como el módulo Main se encuentran disponibles en Moodle.

4. Fecha de entrega y entregables

Semana del 30 de Septiembre al 4 de Octubre de 2019. La entrega debe realizarse antes del comienzo de la clase de prácticas correspondiente. Se deberá entregar un fichero comprimido .zip con nombre **`FAAINTRO_<grupo>_<pareja>.zip`** (ejemplo `FAAINTRO_1461_1.zip`) y el siguiente contenido:

1. Código Python (**Datos.py**) con la implementación de la clase Datos

Esta práctica se evalúa como APTO/NO APTO y no es necesario incluir ningún Jupyter Notebook,