|  |
| --- |
| Universidad Nacional Autónoma de México  Facultad de Ingeniería  División de Ingeniería Eléctrica  Aprendizaje grupo 2  Trabajo: implementación de algoritmo ID3 sobre conjuntos de datos |

Objetivo:

Implementar el algoritmo ID3 visto en clase con las características propias de dicho algoritmo, el alumno comprenderá el funcionamiento del algoritmo en un enfoque práctico y así como su implementación con alguna librería disponible en la web.

Introducción:

En el presente trabajo se desarrolla la posibilidad de implementar un algoritmo de random forest, ID3, con librerías o de forma totalmente propia; que en comparación al proceso mostrado por el profesor, cuando se trabaja con un conjunto de datos mal clasificados o faltos de características únicas se torna en una tarea bastante complicada; esto hace notar un dicho maso menos famoso en el mundo de la minería de datos: el científico de datos se pasa 80% del tiempo depurando la información y haciéndola usable y un 20% programando un algoritmo, testearlo y comprobando resultados.

Desarrollo:

1. En el caso de la implementación propia del árbol de decisión, pese a que Python es un lenguaje ampliamente usado en la inteligencia artificial y el Data Mining, encontré escasamente practico la implementación del algoritmo sobre este lenguaje, pues al calcular las entropías y comparar valores entre si daba la posibilidad de encontrar incoherencias y resultar en muchos errores indeseables y en ocasiones difíciles de tratar.

Por lo que opte por la implementación en un lenguaje bastante poco conocido (lua) con el objetivo de simplificarme aún más la vida respecto a la programación del algoritmo.

El resultado de esto es un programa bastante estable y rápido que es capaz de implementar el algoritmo ID3 sobre un conjunto variable de datos, aprovechando las características inherentes de los programas implementados en lua, entre las que podemos destacar: rapidez, sencillez y posibilidad a encontrar errores de forma mucho más eficiente o rápida que en un programa en Python.

Como dato importante a mencionar es que lua solo cuenta con una única estructura de datos, la tabla hash, la cual es abusada y deformada para implementar todos y cada uno de las posibles estructuras de datos conocidas.

La única posible limitante con la que me pude haber visto enfrentado en lua respecto a Python es la necesidad de implementar mi propia función de logaritmo para n base, que no fue más que una división de logaritmos.

Talvez con un poco más de tiempo incluso pudiendo implementar una versión bastante decente de la librería **sklearn.tree** de Python, con la posibilidad de exportar el grafo a **graphviz** sin mayor complicación.

Las funciones que podemos encontrar en mi algoritmo, fuera de las necesarias para poder cargar un archivo json en memoria son:

* Logn(numero,base): calcula el logaritmo de el numero en una base dada
* Entropy\_element(value,total): calcula el valor individual de la entropía de un dato, que puede ser el valor positivo o negativo, función que simplifica la labor de escribir la función de entropía.
* Entropy(x,y): calcula la entropía dado los datos positivos y negativos(x e y)
* Ganancia\_element(x,y,total): calcula la ganancia individual de cada elemento sobre los posibles valores de una columna.
* Ganancia\_element\_condition(ldata,lfit,lpos\_value,lneg\_value,total,lconditions): calcula na ganancia conjunta de un valor de una columna respecto de una serie de restricciones (lconditions).
* Ganancia\_condition(ldata,lfit\_column,lpos\_value,lneg\_value,column,table\_values,lconditions): calcula la ganancia conjunta de una columna dadas una lista de restricciones.
* Count\_value\_condition(ldata,lconditions):cuenta el número de valores que cumplen con las condiciones.
* Count\_posible\_table\_values(table\_data): cuenta todos los posibles valores que están contenidos en el conjunto de datos leído por columna.
* Table.clone(arg): copia el valor de una tabla sobre otra
* subID3(ldata,column\_fit,positive\_value,negative,value,valores\_tabla,condiciones,valores,verificados):la función implementa una subversión del algoritmo id3 con una serie de restricciones iniciales además de llevar la relación de los valores de columna ya visitados sobre la rama, con el objetivo de eliminar la redundancia. La función sirve para determinar todos los valores de forma recursiva.
* Get\_tree\_representation(positive\_value,negative\_value,original,values): la función obtiene la salida deseada para el ejercicio.
* Inspect\_tree(original\_values): la función tiene el objetivo de imprimir todas las rutas hacia las hojas del árbol de forma decendente.
* Inspect\_leafs(hojas,original\_values): la función tiene como objetivo obtener un conjunto de hojas junto con sus rutas e imprimir en consola las condiciones necesarias para llegar a estas.
* ID3(ldata\_column,column\_fit,positive\_value,negative\_value,original\_value): la función comienza calculando el nodo raíz del algoritmo id3 y manda a llamar a subid3 con una serie de restricciones.
* Leave\_one\_out(dataset): la función dropea un dato del conjunto de datos leídos del archivo.
* Build\_tree(): la función ordena el conjunto de hojas almacenadas en el algoritmo ID3 de forma descendente, de forma que represente el recorrido en niveles del árbol.
* Eval\_tree(element,original\_values):
* La función obtiene un elemento a evaluar sobre el árbol y recorre el árbol evaluando coincidencias en los calores. Necesita los valores para imprimir de forma adecuada y entendible los resultados.

1. En el caso de la implementación de una librería sobre un conjunto de datos no etiquetados puedo decir que se tornó en una tarea bastante complicada de ejecutar pues la función que supuestamente ayudaba al etiquetado de los valores tendía a dar datos poco validos o que daban poca información sobre lo que ocurría en el árbol, de modo que al finalizar la ejecución de la función “fit” el árbol resultante estaba o sobre entrenado o era invalido. Esto se puede observar al momento de la implementación con de una predicción, la cual es en la mayoría de los casos imposible debido a las características de los datos.

Las librerías necesarias para esta tarea son:

Numpy para las pibrerias de sklear, pandas para leer los datos del archivo csv,

Sklearn.model\_selection.train\_test\_split para subdividir los datos como necesitamos

Sklearn.tree.desiciontreeclasifier: que puede ser substituido por la librería id3.Id3estimator

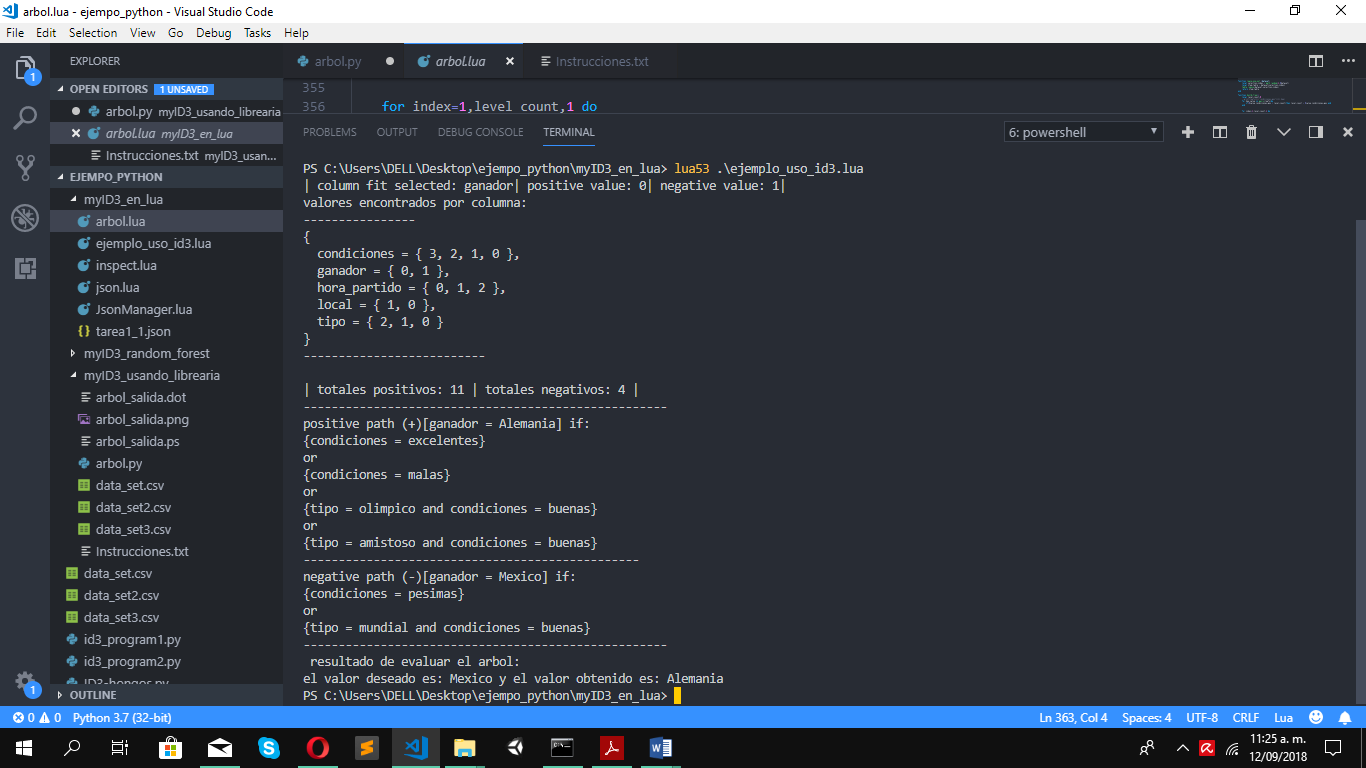
Sklearn.metrics

En este caso no es necesario la programación de ninguna función adicional y las funciones necesarias para poder obtener el árbol son:

* Read\_cvs
* labelencoder.fit\_transform
* train\_test\_split
* id3estimator
* fit
* predict

1. Para el caso de la implementación de un random forest usando las mismas librerías nos encontramos con las mismas complicaciones de forma inherente.

Resultados:

1. Librería propia:
2. Librería de terceros:
   1. Revisar el archivo arbol\_salida.png
   2. Observaciones: debido a la mala clasificación de la función fit\_transform es que salen resultados poco clasificables o totalmente dispersos, que es importante notar que no son tan dispersos como lo serian de haber omitido la omisión del campo ID al principio de los datos.
3. Librería de terceros para ID3 para implementar un random forest:
   1. Revisar el archivo arbol\_de\_forest.png
   2. Se puede observar fácilmente que la dispersión y dimensión de los datos contribuye a que en comparación al inciso anterior, al árbol obtenido sea considerablemente más pequeño.

Conclusiones:

La implementación de librerías suele ser una de las opciones más utilizadas con la única limitante de que es necesario conocerlas a fondo para poder evitar los posibles problemas que estas podrían generar.

Para casos específicos bien podría ser más redituable y practico el generar una librería propia con la ventaja de poder tener total control de las características de esta.

Fuera de esto, respecto a la implementación exitosa del árbol por parte del mi programa, que aunque no cuenta con una salida exactamente igual a la descrita en el documento de especificaciones del trabajo, la salida cuenta con características definibles fácilmente por una gramática y por lo tanto procesable por un programa para generar un árbol. E incluso en caso contrario, la traducción del árbol de decisiones implementado a un archivo .dot para graphviz es relativamente simple.