Predicción de la Diabetes con Algoritmo ID3

Hecho por Carlos Salas Alarcón con la asistencia del profesor universitario, Diego Viejo Hernando.

| Repositorio de GitHub : https://github.com/csalas-alarcon/id3-diabetes |

1. Introducción

El objetivo de este proyecto ha sido entrenar un modelo de Aprendizaje Automático que sea capaz de diagnosticar o no diabetes en los pacientes. Se trabajó con una base de datos con información de 100.000 sujetos y su diagnóstico, además de 28 parámetros con los que después nos hemos ayudado para entrenar el modelo.

El modelo que hemos decido implementar es el algoritmo ID3 (Iterative Dichotomiser 3) el cual es un árbol de decisión que se usa para procesos de categorización como es el propuesto. Se han hecho dos versiones de dicho algoritmo, un ID3 puro, en el cual cada rama se guardan todas las combinaciones posibles, y otro con una técninca de pre-poda que obliga al modelo a generalizar la información.

2. Resultados

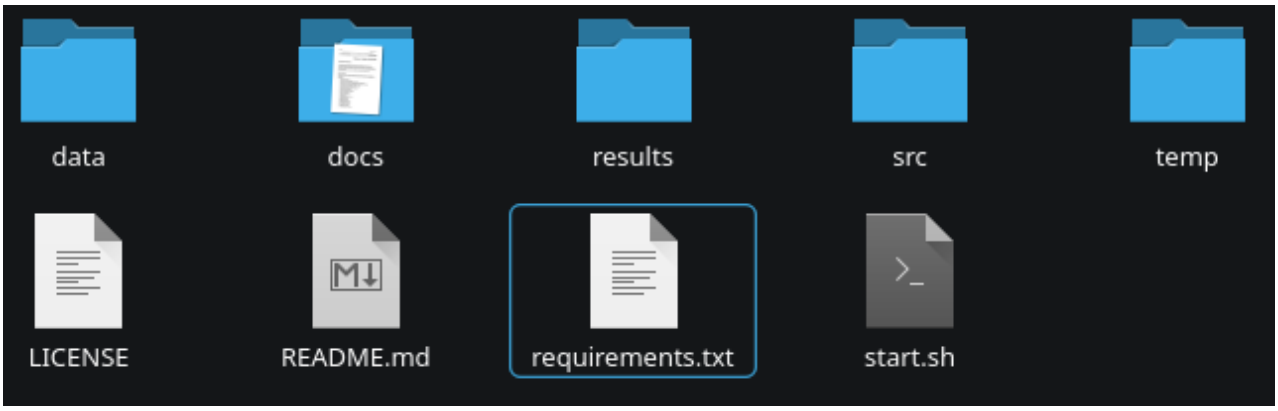
Después de muchísimo esfuerzo, dificultades y contratiempos, el entrenamiento del modelo fue, si se puede decir, un éxito. Desde las primeras versiones funcionales a la última se pasó de un 13% a un 91% de precisión.

Esto se consiguió gracias a técnicas avanzadas como categorización dinámica de las variables continuas, selección exclusiva de los parámetros más prometedores, una selección de un buen criterio de poda, la mezcla de instancias en la base de datos para quitar sesgos iniciales en nuestra base de datos y muchísimos otros pequeñas adiciones al proyecto.

3. Estructuración del Contenido

1. Introducción
2. Resultados
3. Estructuración
4. Organización del Proyecto
5. Bases Matemáticas
6. Flujo de la Información
7. Conclusiones

4. Organización del Proyecto

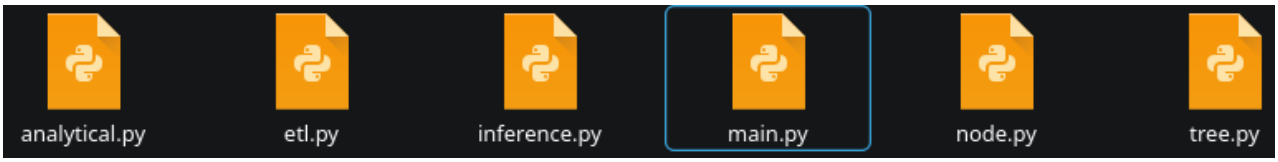


El proyecto tiene 3 carpetas principales: data/, la que contiene las bases de datos en formato CSV.; src/, donde se encuentra todo el código de Python; y results/ que es donde se guardan los intervalos de las variables continuas, las variables con mayor ganancia de información y el modelo ya entrenado, todo en formato JSON.

Además de eso también está la carpeta docs/, en la que está el enunciado de la práctica y este mismo informe, y la carpeta temp/ que es una carpeta intermedia donde se guarda la base de datos preprocesada con los datos continuos ya categorizados. Además por supuesto tenemos la licencia GLIB-3, un README.md, un requirements.txt donde se especifican las librerias a importar (pandas, numpy, scikit-learn) y, por último, un guión de Bash con el que se crea el entorno virtual y se ejecuta el código.

Cabe mencionar que la carpeta temporal temp/ se borra después de la ejecución del entrenamiento y pruebas de validación. Los parámetros ,como división de la base de datos o número de categorizaciones, solo se puede cambiar desde el código.

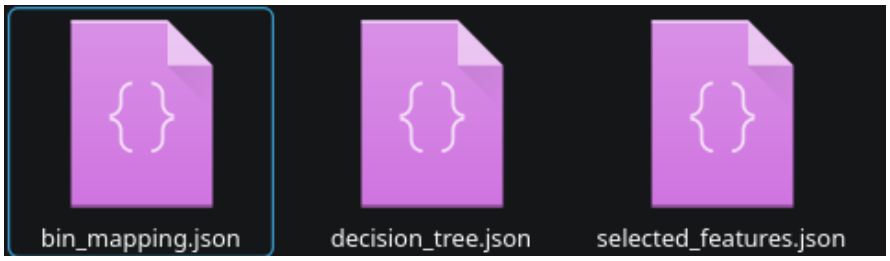
4. Organización del Proyecto - Python



La carpeta más compleja es src/, sus contenidos se pueden ver en la imagen. En orden alfabético, el funcionamiento y propósito de los archivos es el siguiente:

- analytical.py - Optimiza la categorización de variables continuas usando una versión unidimensional del árbol ID3 con prepoda del archivo tree.py. También, calcula la ganancia de información para cada variable y escoge las 5 mejores. Produce: bin_mapping.json y selected_features.json
- etl.py - Se encarga de los procesos de extracción, transformación y carga del proyecto. Tiene tres métodos principales: load el cual mezcla la base de datos, categoriza variables y recorta la información de los parametros seleccionados; node_to_dict, el cual guarda el modelo recien entrenado en decision_tree.json; y, dict_to_node, el cual reconstruye el modelo a partir del JSON guardado.
- inference.py - Donde se declara la clase Engine, la cual infiere los resultados y después los valida.
- main.py - Es el script principal y el que llama al resto de funciones para al final imprimir la puntuación que tiene el algoritmo para cada clase.
- node.py - Archivo que desribe la clase Node, la cual representa cada uno de los nodos del árbol de Decisión.
- tree.py - Archivo que describe las clases DecisionTree y DecisionTreePruning, versiones del algoritmo ID3 que entrenan el módelo.

4. Organización del Proyecto - JSON



La carpeta intermedia más importante es results/, en la cual se encuentran tres archivos JSON muy importantes. bin_mapping.json y selected_features.json son ambos creados por analytical.py y consumidos por etl.py. Estos dos archivos se encargan de guardar los intervalos de las variables continuas y de guardar los archivos seleccionados respectivamente. Por útlimo, decision_tree.json es el mapeado de los nodos del árbol de decisión en un formato JSON. Este archivo es creado por tree.py y consumido por inference.py.

5. Bases Matemáticas del Proyecto - Limitaciones

El algoritmo ID3 es un modelo supervisado de aprendizaje automático usado principalmente en tareas de categorización como la propuesta para este proyecto. El funcionamiento se basa en ramificar la rama (o tronco) de un árbol de decisión con tantas subramas como valores únicos tiene cada parámetro tantas veces como parámetros haya de manera recursiva. Por cuestiones de precisión y eficiencia se suelen usar parámetros de poda.

La principal limitación de este tipo de modelos es la cantidad de datos que requieren. Por ejemplo, tenemos 28 parámetros y asumiendo una media de 5 valores válidos por parámetro, aplicando la siguiente formula $V^p = 5^{28}$ podemos ver que hace falta una base de datos de 372 quintillones de filas para entrenar completamente a nuestro modelo.

5. Bases Matemáticas del Proyecto - Teoría de la Información

Para hacer un algoritmo inteligente hicimos que en cada ejecución recursiva del árbol durante su entrenamiento escogiera el parámetro con la mayor ganancia de información según nos dice la Teoría de la Información de Shannon. La fórmula que usamos para la ganancia de la información es la siguiente:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{S_v}{|S|} \cdot H(S_v)$$

Lo que hace esta fórmula es medir cuanto baja la entropia después de dividir la fórmula según una cierta característica. En esta función de ganancia de información S representa el dataset, A es la característica en cuestión, H() es la entropía, y v es cada valor válido de esa característica. Su implementación en código es la siguiente:

```
return initial_entropy - sum([  # Apply the Info Gain
Formula
    value_freq/ self.length*
    self._calculate_entropy(indices, value)
    for value, value_freq in
    feature_frequencies.items()
])
```

La ganancia de la información como se puede ver depende de otra función, la fuinción de la entropía, la cual, como el nombre indica, mide el desorder que hay dentro del dataset en el que trabaja. Su fórmula es la siguiente, en la que se mantiene la simbología de la fórmula anterior.

$$H(S) = -\sum_{(i=1)}^{n} p_i \cdot log_2(p_i)$$

Su implementación en código es:

```
return sum([ # Apply the Entropy Formula
    -count/ newlength* math.log(count/ newlength, 2)
    if count else 0
    for count in newfrequency.values()
])
```

5. Bases Matemáticas del Proyecto - Análisis

Finalmente, para descubrir cuanta información realmente estaba aportando cada parámetro decidí usar un estadístico de contraste de hipótesis. En concreto usé este estadístico de Chi-Cuadrado:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Esto permitió poder descartar directamente las variables que menos influencia tenían en la clasificacion de los pacientes. La tabla de puntuaciones para cada variable más prueba y error permitió acotar el número óptimo de parámetros a, tan solo, 5. Su implementación en código es:

```
selector = SelectKBest(score_func=chi2, k=k)
selector.fit(X, y)
```

Además es muy, muy importante como se categorizan las variables continuas en el proyecto. Se consiguen categorizar de una manera dinámica y óptima a través de una versión unidimensional de nuestro propio árbol de decisión.

Se dividen por percentiles todos los valores continuos de un cierto parámetro y todos estos valores nuevos que hacen de barrera se alimentan a un árbol de datos unidimensional. El árbol en cada recursión calculará cuales de estas divisiones se relaciona más con los diagnósticos y guardará las divisiones más relevanes eliminando todas las demás una vez ocurra el criterio de poda. Nuestro criterio de poda es que cualquier nodo decisión necesita tener un número determinado de filas en las que basarse. La implementación en código ocurre aquí:

```
# 2. Use PRUNING tree to find the bins
tree_finder = DecisionTreePruning(temp_df,
[feature_name])

# We set a threshold so it only keeps 'meaningful' bins
tree_finder.min_samples = 25

# 3. Train
root = tree_finder._training(None, [0], None)
```

6. Flujo de Información

El proceso cronógico de la información es el siguiente:

1. Se ejecuta el script analytical.py:
    a. Para cada columna continua se generan categorias (o bins) óptimas llamando a un árbol de decisión unidimensional.
    b. Se categorizan todos los valores continuos y se guarda.
    c. Se lee la nueva base de datos y se aplica el estadístico Chi-Cuadrado para seleccionar y guardar los mejores parámetros.

2. Se ejecuta main.py:
    a. Se carga la nueva base de datos.
        i. Mezcla el dataset.
        ii. Codifica todos los valores
        iii. Lo divide en entrenamiento y validación.
    b. Se instancia el árbol de decisión
        i. Se le establecen parámetros
        ii. Se ejecuta
        iii. Guarda la información en un archivo JSON
    c. Se instancia el motor de inferencia
        i. Se reconstruye el árbol.
        ii. Se obtienen los diagnósticos
        iii. Se validan y se imprime el informe.

6. Conclusión

Este proyecto ha sido muy complejo y me ha forzado a entender e implementar muchas cosas nuevas, pero precisamente por eso me ha encantado. Además, ahora sé que puedo hacer IA que ayuden a la gente. Estoy muy orgulloso de este proyecto y espero hacer más como este en el futuro.

Carlos Salas Alarcón