Introducción a la Inteligencia Artificial - 2019
UTN.BA - Extensión Universitaria

Integrantes del grupo: Gerardo Ariel Lopez, Javier Ayunta Maquera, Rubén Matías Violi.

IMPLEMENTACIÓN DE UN SISTEMA INTELIGENTE

I - PROBLEMA

Definición del problema:

Predecir la actitud de compra de los visitantes al sitio de *Google Merchandise Store*, la tienda de comercio electrónico que vende productos de la marca Google, mediante un Sistema Inteligente.

El objetivo es determinar si un visitante hará o no una compra en función de sus patrones de navegación por el sitio, datos geográficos y otros atributos.

Identificación de las fuentes de información:

La motivación del trabajo asignado surge a partir del dataset disponible en el sitio *Kaggle* denominado Google Analytics Sample (BigQuery). Ésta es una base de datos pública perteneciente a *Google Analytics 360*. Son datos reales de fuentes de tráfico, de contenido y transaccionales del sitio *Google Merchandise Store*.

II - DATOS

Recolección de los datos:

La información se importó de *Google Cloud*. Según informa Google en su Esquema de BigQuery Export. El dataset está compuesto de tablas que contienen transacciones correspondientes a visitas de usuarios de todo el mundo durante un año (desde el 01/08/2016 al 01/08/2017) cada tabla contiene un solo día.

En la tabla, cada fila corresponde a un evento durante la visita del usuario al sitio y contiene cientos de atributos y medidas (por ejemplo hacer click sobre un producto, ver detalle del producto, agregar al carrito de compras, quitar del carrito, comprar, pagar, geolocalización, ISP, etc).

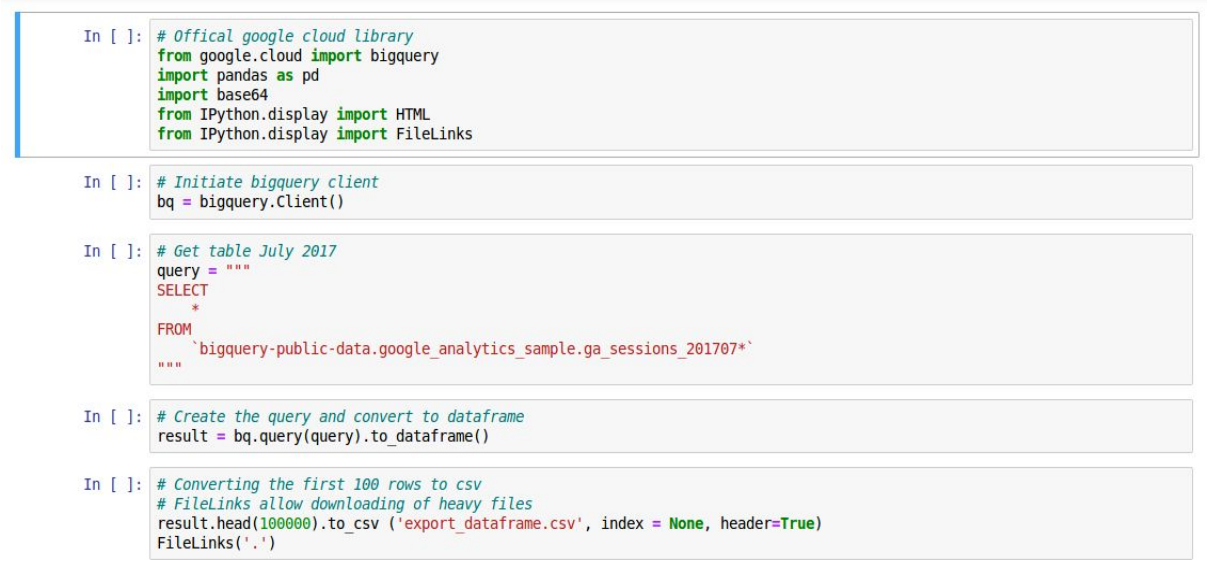
En el proceso de recolección de los datos se hizo evidente que no sería posible utilizar el año entero de información, ni siquiera varios meses. Esto debido a la excesiva cantidad de

tiempo que empleaba el proceso de importación de los datos (y también a fallas en el intento de descarga de este archivo de gran volumen).

Finalmente se decidió limitar la descarga a un sólo un mes (julio de 2017).

Extracción de los datos:

En *Kaggle* se creó un *notebook* empleando *Python* para la extracción de datos desde *Google Cloud*. Esto resultó en un archivo de 1.6 GB llamado `export_dataframe.csv`
Nombre del archivo para extraer los datos: `Dataset Julio 2017.ipynb`



```
In [ ]: # Offical google cloud library
from google.cloud import bigquery
import pandas as pd
import base64
from IPython.display import HTML
from IPython.display import FileLinks
```

```
In [ ]: # Initiate bigquery client
bq = bigquery.Client()
```

```
In [ ]: # Get table July 2017
query = """
SELECT
    *
FROM
    `bigquery-public-data.google_analytics_sample.ga_sessions_201707*`
"""
```

```
In [ ]: # Create the query and convert to dataframe
result = bq.query(query).to_dataframe()
```

```
In [ ]: # Converting the first 100 rows to csv
# FileLinks allow downloading of heavy files
result.head(100000).to_csv ('export_dataframe.csv', index = None, header=True)
FileLinks('.')
```

Estudio y preparación de los datos:

Se concluyó que los datos de todo el mes debían descargarse en una única tabla para facilitar su administración y uso. Una vez obtenida la tabla correspondiente a la actividad de julio de 2017, al visualizarla, se encontró que los mismos presentan restricciones de privacidad. Esto es porque se trata de una versión pública donde numerosos atributos fueron anulados (las columnas afectadas están etiquetadas explícitamente con esta declaración). Además el tamaño de la tabla no hacía posible su procesamiento en una PC. Por lo tanto hubo que hacer una limpieza de los datos que incluyó al mismo tiempo filtrar qué atributos eran relevantes para el objetivo del problema. Se seleccionaron los siguientes atributos como representativos para cumplir con el objetivo y se creó el atributo "*BuyFlag*":

- `visitNumber`
- `channelGrouping`
- `visits`
- `pageviews`
- `timeOnSite`
- `newVisits`
- `BuyFlag`
- `sessionQualityDim`

- `browser`
- `operatingSystem`
- `isMobile`
- `continent`
- `subContinent`
- `country`
- `networkDomain`
- `action_type`

Para este proceso de limpieza y filtrado se utilizó la librería de Python *pandas* en *Jupyter Notebook*, *MS Excel* trabajando directamente sobre las columnas y también se utilizó *RapidMiner*(*Turboprep*) pero sólo a los efectos de conocer el uso de la herramienta.
Nombre del archivo: `Transformacion.ipynb`
El archivo de salida final es: `Datos.xlsx`

El archivo resultante es una nueva tabla con menor volumen de información y accesible a la memoria y capacidad de cómputo de una computadora personal.

Luego del formateo de esta tabla y a partir de un nuevo análisis visual se encuentra que los datos podían generar un sesgo debido a que la proporción de visitantes que compran es mínima con respecto a los que no compran. Por lo tanto se procedió balancearlos antes de la aplicación de los modelos.

III - TECNOLOGÍA

Se desea un sistema que aprenda automáticamente a partir de datos disponibles. La tecnología seleccionada son algoritmos de Machine Learning (Regresión logística y Deep Learning). La estrategia de aprendizaje será por aprendizaje supervisado (tenemos atributos de entrada y de salida).

La tarea que realizará el sistema es predecir una conducta por lo tanto la familia de arquitecturas a utilizar serán RNA & Deep Learning.

IV - MODELO

Los modelos seleccionados dentro de la familia de arquitecturas para resolver el problema son:

- Regresión logística (en *Python* usando el entorno *Jupyter notebook*)
- Deep Learning (en *RapidMiner*)

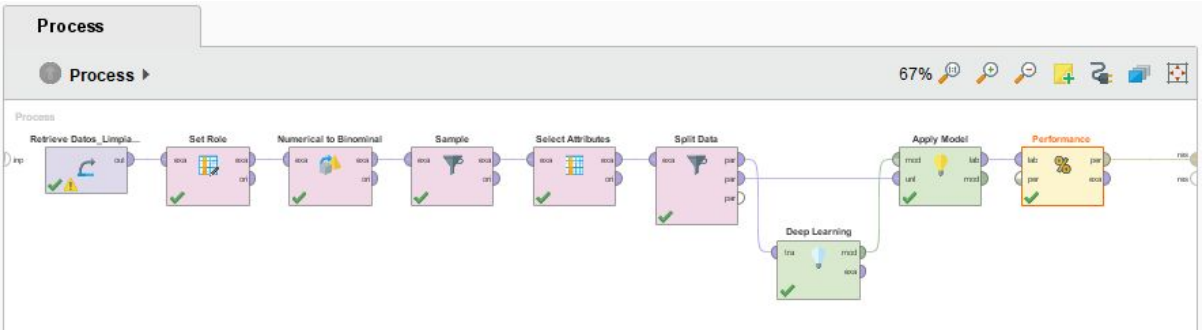
Para construir los modelos se deben separar los datos de prueba de los datos de entrenamiento/validación. Las métricas de evaluación por tratarse de algoritmos de aprendizaje supervisado serán la matriz de confusión y la curva ROC.

REGRESIÓN LOGÍSTICA

Para el tratamiento, categorización y visualización de información se utilizaron las librerías de Python (*pandas*, *matplotlib, sklearn*).
Algoritmo, entrenamiento, prueba y resultados: `Probabilidad_Compra.ipynb`
Datos de entrenamiento/validación: 75%
Datos de prueba: 25%

A partir de la observación de la curva ROC se concluye que el modelo es eficiente para predecir y por lo tanto es capaz de generalizar.

DEEP LEARNING
Algoritmo, entrenamiento, prueba y resultados: `Deep Learning.rmp`
Datos de entrenamiento/validación: 75%
Datos de prueba: 25%



Operadores utilizados:
Retrieve data → *Set Role* (para señalar como label a *BuyFlag*) → *Numerical to Binomial* (para pasar el valor 0-1 de *BuyFlag* a *True-False*) → *Sample* (para balancear los datos) → *Select Attributes* (para seleccionar los atributos que consideramos serán útiles al objetivo) → *Split Data* (para separar en datos de entrenamiento/validación y datos de prueba) → *Deep Learning* (Ejecuta el algoritmo de deep learning) → *Apply Model* (para aplicar el modelo entrenado en los datos de prueba) → *Performance* (para obtener la matriz de confusión y la curva ROC)

accuracy: 99.92%

	true false	true true	class precision
pred. false	928	1	99.89%
pred. true	0	308	100.00%
class recall	100.00%	99.68%	

V - Problemas e inconvenientes encontrados durante la realización del trabajo práctico:

- Organización de los datos:

Fue necesario interiorizarse acerca del significado y organización de los datos. Para ello estudiamos el Esquema de Datos provisto por Google (*BigQuery Export schema*). Allí encontramos que:

- **Tabla**: dentro del dataset, cada tabla representa un día completo y se importa con el formato "`ga_sessions_YYYYMMDD`".

- **Filas**: cada fila representa una sesión en *Analytics 360* (la plataforma de marketing de Google).

- **Columnas**: nos encontramos con 15 columnas donde 6 atributos son de tipo RECORD. A su vez, en algunos de ellos, también se anidan más atributos de este tipo, totalizando un total de 30. Por lo tanto, la tabla contiene datos anidados que, desglosados, nos dan mas de 300 atributos tipo STRING, INTEGER y BOOLEAN.

- Atributos anidados:

  Si bien este dataset contiene tablas con 15 atributos, el grupo se encontró con columnas conteniendo a su vez múltiples atributos (datos anidados de tipo RECORD). Esto implicó un tiempo de aprendizaje para entender cómo importarlos, visualizarlos y procesarlos usando los comandos correspondientes (en *Python*), lo cual a su vez implicó interiorizarse en el uso de código específico.

- Significado de los datos:

  Las descripciones provistas por Google para cada campo, resultaron ser muy pobres, aunque nos dió una idea básica del significado. Así y todo, fue insuficiente y debimos analizar manualmente los datos en *MS Excel*, creando tablas dinámicas, filtros, etc. Gracias a este análisis, descubrimos que:

  1. Algunas columnas estaban en NULL (sin valor) o un valor constante para todos los registros: Descartamos esas columnas para nuestro trabajo.
  2. Columnas parcialmente incompletas: Por ejemplo, el atributo "`hits.eCommerceAction.action_type`" que nos hubiera resultado de gran utilidad para determinar, entre otras cosas, si un cliente compró o no, resultó que no era de fiar, dado que no había correlación con el campo "`totalTransactionRevenue`".

- Volumen de datos:

  El plan original implicaba tomar 1 (un) año completo de información.
  El primer escollo que nos encontramos fue que *Kaggle* permitía bajar la información de a un archivo por día. Esto implicaba que luego debíamos unir las tablas para poder tener en una sola los datos de semanas o meses. Eso lo pudimos resolver y

pudimos obtener la información en una única tabla mediante un código en *Python* bajando la información directamente desde *Google Cloud*.
Nos dimos cuenta que descargar un año de información empleaba un tiempo excesivo, lo mismo ocurría si se intentaba con varios meses
Finalmente decidimos descargar y trabajar con 1 sólo mes.
No obstante ello, el archivo generado para un sólo mes tenía un tamaño de 1,6 GB, lo cual resultó inmanejable para las aplicaciones (fracasamos en varios intentos) razón por la cual debimos proceder a la limpieza de los datos.

- *BuyFlag*:

  Al no contar con un campo que nos indique si el cliente compro o no, luego del análisis de los datos determinamos que podíamos resolverlo agregando una nueva columna (la llamamos *BuyFlag*) a nuestro archivo con una fórmula en *MS Excel* con la siguiente lógica:

  ```
  IF (totalTransactionRevenue > 0; 1; 0)
  ```

  donde, se devuelve 1 si el visitante compró y 0 si no compró.

- Balanceo de Datos

  Descubrimos que la proporción de sesiones que significaron una compra es extremadamente baja respecto a las visitas totales: menos del 2%, por lo tanto al momento de aplicar el modelo tendríamos un problema de *overfitting* (sesgo hacia la mayor cantidad de compras = 0).

  Para que el modelo pueda identificar, discriminar mejor las clases y eliminar el *overfitting* es necesario tener casi las mismas cantidades de fila en la variable *BuyFlag* para los casos de 0 y 1 . Eso nos llevó a balancear los datos de tal manera que el algoritmo entienda y clasifique mejor.

  Para este ejercicio en particular tuvimos 87.3% de filas de la categoría 0 y 12.7 % para la categoría 1.

VI - Conclusiones generales de la entrega:

1.- Es evidente que existen muchos algoritmos supervisados para entender el comportamiento y realizar la probabilidad de compra de un producto, para este caso en particular se buscó el más sencillo y fácil de entender. Para un modelo productivo fácilmente puede ser adaptable y escalable, siempre y cuando se respeten los supuestos y requisitos de este algoritmo.

2.- La aplicación de soluciones como ésta, le da herramientas a las áreas de Marketing y Ventas, de segmentar sus políticas de promociones por Zonas

geográficas, grupos etarios, días y horarios, etc. También poder analizar más en profundidad a los no-compradores para lograr revertir esa situación.
