# Proyecto Final Inteligencia Artificial

**Flores González, Jesús Eduardo.** *

* *Universidad Nacional Autónoma de México, Facultad de Ingeniería,*
*e-mail: jef007.flores@gmail.com*

**Resumen**: El presente documento muestra la documentación y explicación del proyecto analizador de datos para la materia de Inteligencia Artificial impartida en la Facultad de Ingeniería de la UNAM, en el se abordan los objetivos del proyecto y se describe sobre el uso de las tecnología implementadas junto con el desarrollo de esta software para computadora. El como surge el diseño de la interfaz planteada así como una explicación de los aspectos mas importantes del código.

**Palabras clave**: Inteligencia artificial, algoritmo, python, apriori, clustering, correlacional, métricas de similitud, regresión logística, interfaz, pandas, matplotlib, software, PyQt5.

## 1. OBJETIVO

Diseñar software que por medio de una interfaz se puedan analizar datos donde el usuario proporcione un archivo con dicha información y estos puedan ser analizados mediante los algoritmos de inteligencia artificial vistos en clase poniendo en práctica los conocimiento adquiridos en la prácticas así como conocimientos que se han adquirido a lo largo de la carrera de Ingeniería en Computación.

## 2. REQUERIMIENTOS

- Carga de archivos proporcionados por el usuario en formatos .txt, .csv, y .xls.
- Visualización de los datos en la interfaz.
- Poder seleccionar el algoritmo que se desea emplear para poder analizar los datos.
- Mostrar las salidas que puedan generar los diversos algoritmos (texto y gráficos en caso de ser necesario)
- Poder seleccionar parámetros de entrada para los algoritmos con el fin de tener un análisis personalizado al gusto del usuario

## 3. HERRAMIENTAS DE DESARROLLO

Lenguaje de alto nivel Python en su versión 3.8.7 el cual es un lenguaje interpretado el cual se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

Otra herramienta utilizada es PyQt5 junto con Qt Designer el cual es un binding de la biblioteca gráfica Qt para el lenguaje de programación Python. La biblioteca está desarrollada por la firma británica Riverbank Computing y está disponible para Windows, GNU/Linux y Mac OS X bajo diferentes licencias.

## 4. REQUISITOS DE INSTALACIÓN

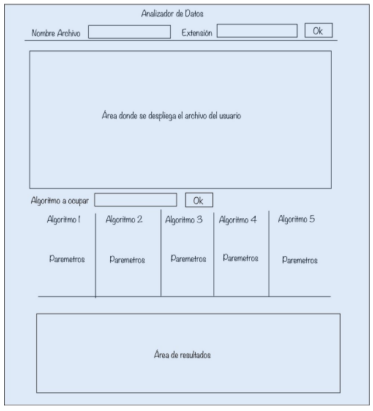
- Python 3.x
- Modulos de Python
  - Sys
  - Pandas
  - Numpy
  - Math
  - Matplotlib
  - Seaborn
  - Scipy
  - Axes3D
  - Kmeans
  - PyQt5
  - Apyori
  - Kneed
- Sistema Operativo
  - Windows 7, 8, 8.1 y 10
  - Ubuntu 14.04, 16.04 y 16.10 (y derivados)
  - Mac OS X 10.7 (Lion) o superior

Se recomienda de un mínimo de de 6Gb RAM y un procesador de 4 núcleos (QuadCore) ya que es el equipo donde se desarrollo y no se presentaban problemas de velocidad para ejecutar el software.

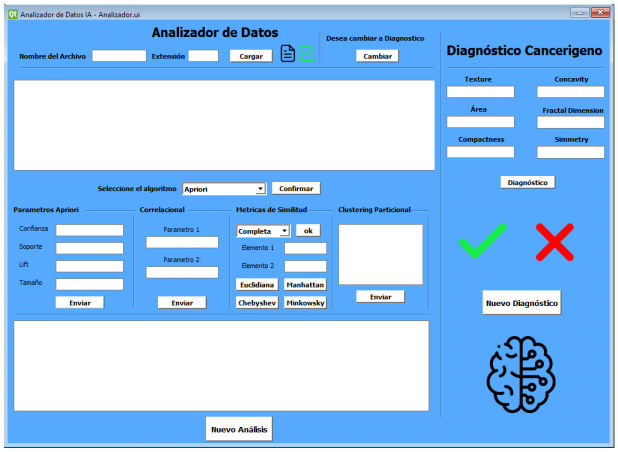
## 5. SOBRE LA INTERFAZ

Para la aplicación se diseño una ventana en la cual se pudiera cargar una archivo especificando su nombre y extensión, además que en esta ventana se pudiera seleccionar una algoritmo de trabajo con el cual analizar los datos previamente cargados.

A continuación se muestra un borrados de una versión temprana del desarrollo.



Posteriormente conforme el desarrollo fue avanzando y se tenían todos los algoritmos a implementar la interfaz final por la cual se opto fue la siguiente.

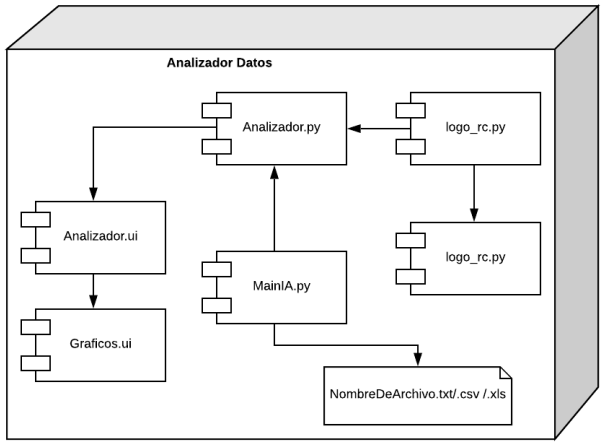



En dicha interfaz podemos observar elementos como:

- Cajas de texto: Las cuales están encargadas de recibir los datos introducidos por el usuario.
- Botones: Los cuales generar eventos al presionarse dichos eventos mandan a llamar a métodos implementados en el código dichos métodos van desde selección de algoritmos, el control de la interfaz y la ejecución de los algoritmos.
- Áreas de texto: Las cuales sirven para mostrar los resultados de los algoritmos como matrices, reglas de asociación, cálculos de distancias entre otras.
- Etiquetas: Se emplean para mostrar texto que no cambia es decir el que muestra la petición de datos, títulos así como imágenes que pueden encontrarse activadas o desactivadas dependiendo de la situación
- Gráficos: Estos elementos permiten que los datos de salida sean mas claros de visualizar aparte de las matrices que se muestran en el áreas de texto.

## 6. DIAGRAMA COMPONENTES

Para el desarrollo del proyecto se creo el siguiente diagrama de componentes UML el cual muestra la interacción de los archivos de código entre si junto con la entrada en este caso un archivo el cual como se especifico en los requerimientos puede contar con extensiones .txt, .csv y .xls.




## 7. FUNCIONAMIENTO DEL CÓDIGO

### *7.1 Lectura de archivos*

Para poder implementar esto un método dentro de la clase AnalizadorDataApp se encarga de obtener el nombre del archivo y extensión cada uno con una caja de texto concatenándolos para así saber que tipo de archivo es el que leerá y entra en un serie de condiciones donde se detecta si es .txt, .csv o .xls.. Una vez dentro de esto se realiza la lectura y se almacena y se muestra en el área de texto 1 para visualización del usuario.

```
#Obtencion del nombre del archivo
  NombreArchivo = self.LE_NombreArchivo.text()
  #Obtencion de la extensi n del archivo
  Extension = self.LE_Extension.text()
  Extension.lower
  global Data
  if Extension == ".txt" :
      File =  NombreArchivo + Extension
      Data = pd.read_table(File)
      DataStr = str(Data)
      self.txt_Texto.setText(DataStr)
```

### *7.2 Selección de algoritmo*

Este método se encarga de detectar el algoritmo con el que se desea trabajar, aparte de detectar esto se encarga de activar y desactivar elementos de la interfaz que no se ocuparan en ese momento así como de empezar a crear algunas paso previos a la ejecución del algoritmo seleccionado. Es importante mencionar que se han omitido las lineas de activación y desactivación sin embargo pueden observar en el repositorio donde se aloja todo el código fuente del proyecto.

```
def Algoritmos(self): #Funci n que controla la
    QComboBox para la selecci n de los distintos
    algoritos implementados
        AlgoritmoSeleccionado = str(self.
            cb_Algoritmo.currentText())
        print(AlgoritmoSeleccionado)

        if AlgoritmoSeleccionado == "Apriori":
            #Se habilitan todas las entradas (LINEAS
                 OMITIDAS)
            global x,y
            x = len(Data.index)
            y = len(Data.columns)
            print(x,y)
            # Se determinan las transacciones
                realizadas
            global Transacciones
            for i in range(0, int(x)):
                Transacciones.append([str(Data.
                    values[i, j]) for j in range(0,
                    int(y))])
            print(Transacciones) #Para ver las
                transacciones ya que tardan en
                consola
        if AlgoritmoSeleccionado == "Correlacional":
            #Se habilitan todas las entradas (LINEAS
                 OMITIDAS)
            Matriz = Data.corr(method='pearson')
            self.txt_Texto_2.insertPlainText(str(
                Matriz) + "\n")
            plt.matshow(Matriz)
```

```
            plt.show()
        if AlgoritmoSeleccionado == "Metricas de Similitud":
            #Se habilitan todas las entradas (LINEAS OMITIDAS)
            self.pB_ok.clicked.connect(self.OptMetricas)

        if AlgoritmoSeleccionado == "Clustering Particional":
            #Se habilitan todas las entradas (LINEAS OMITIDAS)
            #Obtencion del nombre del archivo
            NombreArchivo = self.LE_NombreArchivo.text()
            #Obtencion de la extensi n del archivo
            Extension = self.LE_Extension.text()
            Extension.lower
            global DataCluster
            if Extension == ".csv":
                File = NombreArchivo + Extension
                DataCluster = pd.read_csv(File)
                DataStr = str(Data)
                self.txt_Texto.setText(DataStr)
                self.label_19.setHidden(False) #Se activan iconos de que se ley correctamente el archivo
                #Se habilitan todas las entradas (LINEAS OMITIDAS)
                for column in DataCluster:
                    self.LW_SelecVar.addItem(column)
            if Extension == ".xls":
                File = NombreArchivo + Extension
                DataCluster = pd.read_excel(File)
                DataStr = str(Data)
                self.txt_Texto.setText(DataStr)
                #Se habilitan todas las entradas (LINEAS OMITIDAS)
```

### *7.3 Algoritmo Apriori*

Este algoritmo es utilizado en Inteligencia Artificial y Minería de Datos sobre bases de datos transaccionales, que permite encontrar de forma eficiente çonjuntos de ítems frecuentes", los cuales sirven de base para generar reglas de asociación. En este caso no se trabajo con una conexión una base de datos, esto para que un usuario pudiera cargar los archivos desde una hoja de calculo o un documento plano el cual puede generar desde una base de datos. Una vez que se carga el archivo el algoritmo todavía no entra en ejecución pues necesita de los siguientes parámetros para su funcionamiento:

- Soporte
- Confianza
- Lift
- Tamaño

Una vez introducidos los valores el algoritmo entra en acción mostrando el numero total de reglas generadas junto con cada una de la reglas juntos con los valores calculados para cada una de estas. (Esto puede visualizarse mejor en el funcionamiento del vídeo).

```
def LeerParametrosApriori(self): #Funcion de lectura de los parametros necesarios para poder ejecutar el algoritmo apriorio
    #Se obtienen los parametros de Apriori probenientes de las cajas de texto y sea castean a flotantes
    #En caso de generar un error se muestra un error
    try:
        global Confianza, Soporte, Lift, Tamanio
        Confianza = float(self.LE_Confianza.text())
        Soporte = float(self.LE_Soporte.text())
        Lift = float(self.LE_lift.text())
        Tamanio = float(self.LE_Tamanio.text())
    except:
        QMessageBox.critical(self, "ERROR", "No fue posible realizar ningun calculo, verifique sus datos.")

    #Se generan todas la posiblres reglar con los datos introducidos por el usuario
    Reglas = apriori(Transacciones, min_support=Soporte, min_confidence=Confianza, min_lift=Lift, min_length=Tamanio)
    Resultados = list(Reglas)

    #Se muestra el numero de reglas obtenidas en el area de texto junto con cada una de las reglas obtenidas
    self.txt_Texto_2.insertPlainText("El total de reglas son:" + str(len(Resultados)) + "\n")
    for item in Resultados:
        pair = item[0]
        items = [x for x in pair]
        self.txt_Texto_2.insertPlainText("Regla: " + items[0] + " -> " + items[1] + "\n")
        self.txt_Texto_2.insertPlainText("Soporte: " + str(item[1]) + "\n")
        self.txt_Texto_2.insertPlainText("Confianza: " + str(item[2][0][2]) + "\n")
        self.txt_Texto_2.insertPlainText("Lift: " + str(item[2][0][3]) + "\n")
        self.txt_Texto_2.insertPlainText("==========
        ==========" + "\n")
```

### *7.4 Algoritmo Correccional*

Este algoritmo se encarga de generar una matriz de correlaciones la cual aporta información sobre la relación entre pares de variables obteniendo un subconjunto de variables representativas que no tengan dependencia entre si. Una vez esto los datos se pueden graficar en pares obteniendo una gráfica de correlaciones de la cual se debe hacer una inspección visual para interpretar si los datos tiene correlación junto con el coeficiente de correlación calculado para esos datos que se encuentra en la matriz de correlaciones.

La gráfica de correlaciones se genera después de leer el archivo y esta también ser muestra en un gráfico de colores el cual sirve de apoyo para el usuario.

```
    Matriz = Data.corr(method='pearson')
    self.txt_Texto_2.insertPlainText(str(Matriz) + "\n")
    plt.matshow(Matriz)
```

```
    plt.show()
```

Una vez que se tiene esto en la interfaz se habilita el área de parámetros estos son de los que se obtendrá el gráfico de correlación para inspección del usuario. En caso de que se introduzca de forma errónea el nombre del parámetro o que este no exista en los datos se le notifica un error al usuario.

```
def LeerParametrosCorrelacional(self): #Funci n de lectura de par metros para el m todo de correlaciones donde de obtiene la gr fica de dichos datos
        Parametro1 = self.LE_Parametro.text()
        Parametro2 = self.LE_Parametro_2.text()
        try:
            plt.close() #Limpiar pantalla grafico
            plt.plot(Data[Parametro1], Data[
                Parametro2], 'r+')
            plt.ylabel(Parametro2)
            plt.xlabel(Parametro1)
            plt.show()
        except:
            QMessageBox.critical(self, "ERROR", "El par metro no existe, no es posible graficar")
```

### *7.5 Algoritmo Métricas de Similitud*

Este algoritmo se encarga de calcular la distancia que existen entre datos junto con una matriz de similitud es decir el calculo de todas las distancias. Para esto es necesario generar matrices para cálculos y una donde se almacenara la matriz de similitudes esto es un proceso que se repite para los cuatro tipos de métricas disponibles en este software.

```
DataMetricas = 0
DataMetricas = Data
DataMetricas.drop(0)
x = len(DataMetricas.index)
y = len(DataMetricas.columns)

DataCalc = {}
for i in range(0,y):
    Lista = []
    for j in range(0,y):
        Lista.append('--')
        DataCalc[i] = Lista
        MatrizSimilitudes = pd.DataFrame(data = DataCalc)
         for i in range(0, y):
            for j in range(0, i+1):
                E1 = DataMetricas.iloc[i]
                E2 = DataMetricas.iloc[j]
                Temp1 = []
                Temp2 = []
                for k in range(0, len(E1)):
                    try:
                        Temp1.append(float(E1[k]))
                    except ValueError:
                        pass
                    for k in range(0, len(E2)):
                        try:
                            Temp2.append(float(E2[k]))
                        except ValueError:
                            pass
                #Linea de calculo segun la metrica seleccionada
                MatrizSimilitudes.iloc[i, j] = dist
```

La linea 350 cambia para cada una de las métricas

- Euclidiana

```
    dist = "{0:.05f}".format(math.sqrt(sum((Temp1-Temp2)**2 for Temp1, Temp2 in zip(Temp1, Temp2))))
```

- Manhattan

```
    dist = "{0:.5f}".format(distance.cityblock(Temp1, Temp2))
```

- Minkowsky

```
    dist = "{0:.5f}".format(distance.minkowski(Temp1, Temp2))
```

- Chebyshev

```
    dist = "{0:.5f}".format(distance.chebyshev(Temp1, Temp2))
```

### *7.6 Algoritmo Clustering Particional*

Para la implementación de dicho algoritmo utilizado en la inteligencia artificial aplicada, consiste en la segmentación de grupos de elementos de manera automática que son unidos por características comunes que estos comparten. El método que realiza el funcionamiento del algoritmo genera el gráfico del método del código así como el de los clusters junto con la selección de variables y muestra en el área de texto resultados como los clusters obtenidos.

```
def ClusteringParticional(self):
        VarSelec = self.LW_SelecVar.selectedItems()
        select = []
        row =[]
        for x in range(len(VarSelec)):
            select.append(self.LW_SelecVar.selectedItems()[x].text())
        print(select)

        Matriz = DataCluster.corr(method='pearson')
        VariablesModelo = np.array(DataCluster[select])
        print(VariablesModelo)

        SSE = []
        for i in range(2, 16):
            km = KMeans(n_clusters=i, random_state=0)
            km.fit(VariablesModelo)
            SSE.append(km.inertia_)

        #Se grafica SSE en funci n de k
        plt.figure(figsize=(7, 3))
        plt.plot(range(2, 16), SSE, marker='o')
        plt.xlabel('Cantidad de clusters *k*')
        plt.ylabel('SSE')
```

```
        plt.title('Elbow Method')
        plt.show()

        #Se crean los clusters
        #random_state se utiliza para inicializar el generador interno de numeros aleatorios (mismo resultado)
        MParticional = KMeans(n_clusters=5, random_state=0).fit(VariablesModelo)
        MParticional.predict(VariablesModelo)

        DataCluster['clusterP'] = MParticional.labels_
        self.txt_Texto_2.insertPlainText("Los Clusters son:\n")
        self.txt_Texto_2.insertPlainText(str(DataCluster.groupby(['clusterP'])['clusterP'].count()) + "\n")
        self.txt_Texto_2.insertPlainText("==========================================================" + "\n")

        CentroidesP = MParticional.cluster_centers_

        self.txt_Texto_2.insertPlainText("La matriz de centroides es:\n")
        self.txt_Texto_2.insertPlainText(str(pd.DataFrame(CentroidesP.round(4))) + "\n")

        # Grqfica de los elementos y los centros de los clusters
        plt.rcParams['figure.figsize'] = (7, 3)
        plt.style.use('ggplot')
        colores=['red', 'blue', 'cyan', 'green', 'yellow']
        asignar=[]
        for row in MParticional.labels_:
            asignar.append(colores[row])

        fig = plt.figure()
        ax = Axes3D(fig)
        ax.scatter (VariablesModelo[:, 0], VariablesModelo[:, 1], VariablesModelo[:, 2], marker='o', c=asignar, s=60)
        ax.scatter(CentroidesP[:, 0], CentroidesP[:, 1], CentroidesP[:, 2], marker='*', c=colores, s=1000)
        plt.show()
```

### *7.7 Algoritmo Regresión Logística*

El uso de este algoritmo dentro de la interfaz es para la predicción de si un paciente puede tener un tumor maligno o benigno para esto el modelo se entrena a parte con una cantidad de datos muy grande y una vez que se tiene el modelo de regresión logística este se implanta en la interfaz para poder trabajar directamente con el es decir el modelo no se genera en el programa esto por cuestiones de optimizaciones y ya que solamente se requiere como herramienta de diagnostico. El método encargado en el código para realizar esto es:

```
def LeerParametrosDiagnostico(self): #Funcion que implementa la lectura y ejecucion del diagnostico con los datos propocionados
    global Texture, Area, Compactness, Concavity, FractalDim, Symmetry
    try:
        Texture = float(self.LE_Texture.text())
        Area = float(self.LE_Area.text())
        Compactness = float(self.LE_Compactness.text())
        Concavity = float(self.LE_Concavity.text())
        FractalDim = float(self.LE_FractalDim.text())
        Symmetry = float(self.LE_Symmetry.text())
    except:
        QMessageBox.critical(self, "ERROR", "Verifique los datos introducidos")

    ModeloDeClasificacion = 11.72-0.19*Texture-0.01*Area-2.27*Compactness-3.08*Concavity-0.88*Symmetry-0.21*FractalDim
    Probabilidad = 1/(1+np.exp(ModeloDeClasificacion))

    if(Probabilidad < 0.5):
        #Descativa las cajas y etiquetas donde se introducen los datos
        self.label_13.setHidden(False)
        self.lb_Diagnostico.setText('Tumor benigno')
    else:
        #Descativa las cajas y etiquetas donde se introducen los datos
        self.label_12.setHidden(False)
        self.lb_Diagnostico.setText('Tumor Maligno')
```

### *7.8 Sobre otros apartado dentro del código*

El resto de métodos implementado en el código son para el control de la interfaz, aspectos, orientación de elementos y manejo de errores que puedan surgir.

## 8. CONCLUSIONES DEL PROYECTO

Considero de gran utilidad la herramienta desarrollado durante el curso ya que sirve como un visualizador de datos mediante algoritmos de inteligencia artificial es basten te funcional, sin embargo me gustaría poder optimizar el código haciendo que los algoritmos se ejecuten cuando se tiene la opción en equipo instalado por medio de un GPU a pesar de que en el equipo donde se desarrollo no se presenta ro problemas es cierto que cuando la cantidad de datos es mayor los tiempos de ejecución de los algoritmos penden elevarse. Aparte de esto seria conveniente añadir un barra de progreso cuando se esta ejecutando los algoritmos y estos tarden esto para que el usuario sepa que se están calculando los datos y no crea que la aplicación se ha trabado o genere esa sensación.

## 9. BIBLIOGRAFÍA


Python Org., (2020) ***Python 3.8.7 documentation***. [Online]. Disponible en https://docs.python.org/3.8/

Python Org., (2021) ***PyQt5 Reference Guide***. [Online]. Disponible en ttp://www.latex-project.org/