<https://fracpete.github.io/python-weka-wrapper3/>

**python 3.9.0**

entorno virtual





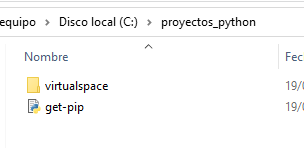


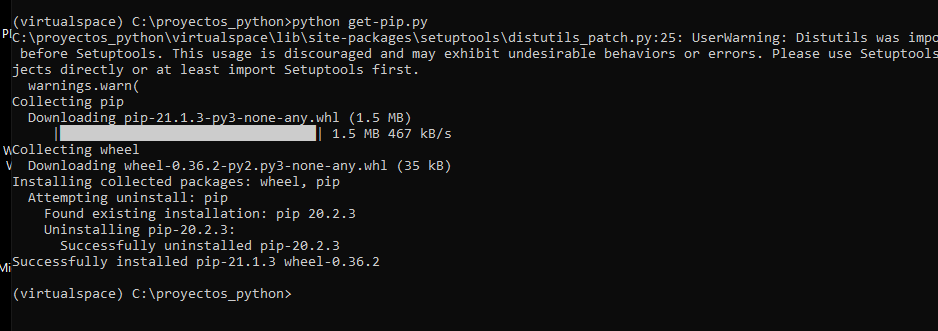
http://fracpete.github.io/python-weka-wrapper3/install.html#windows

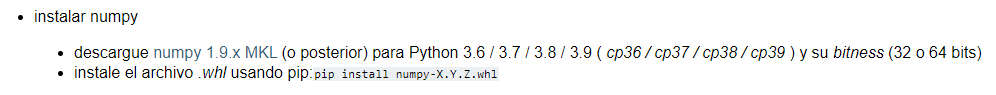


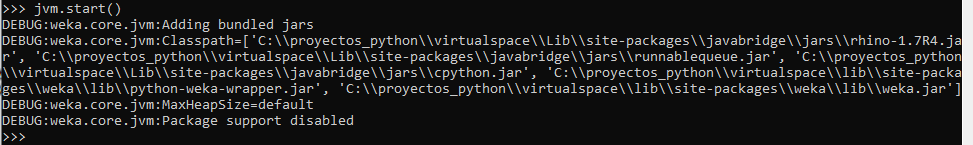
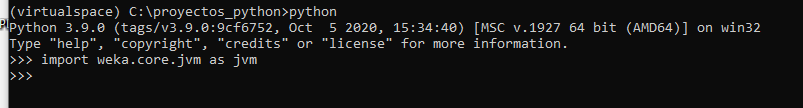
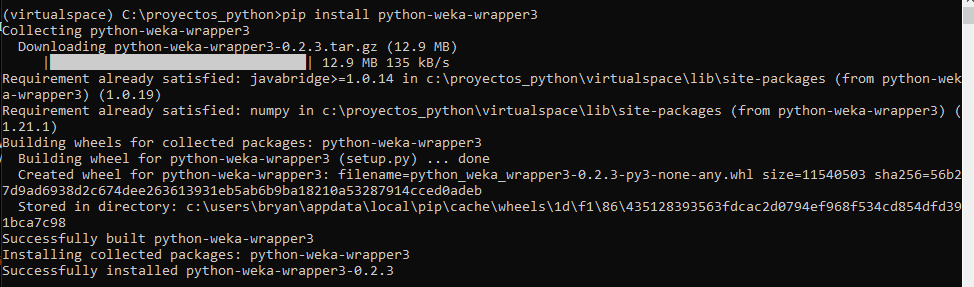
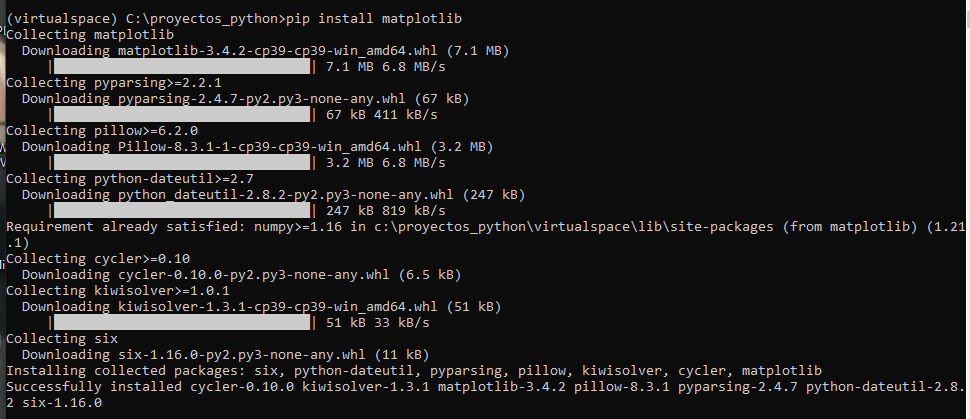
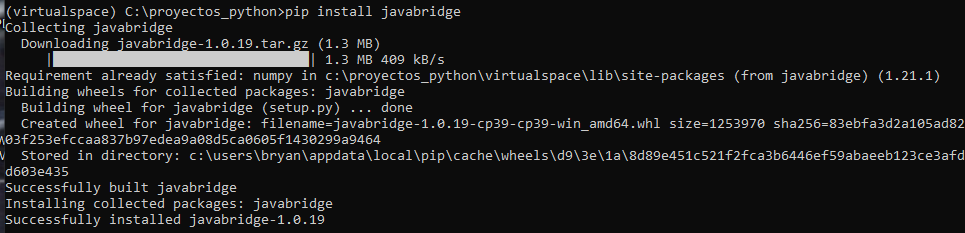
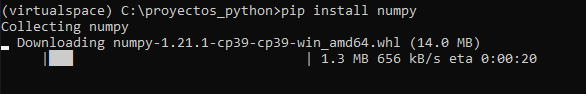




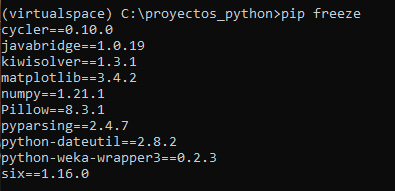


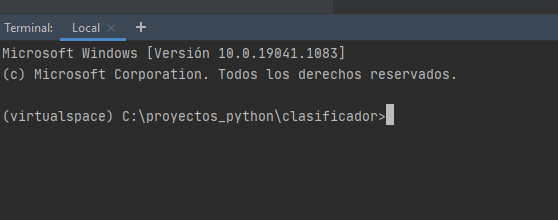
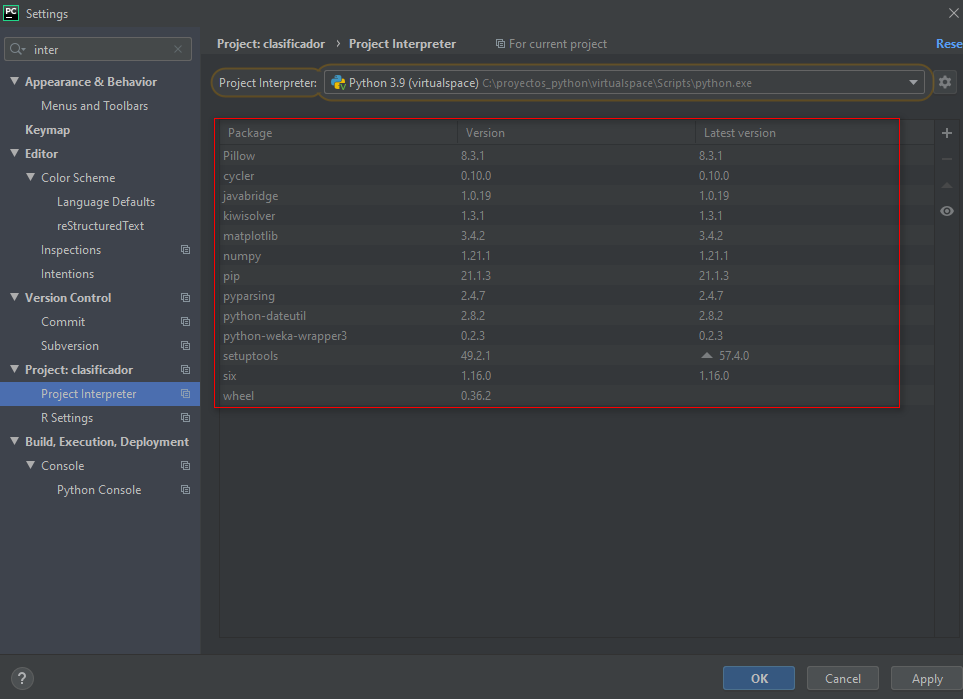
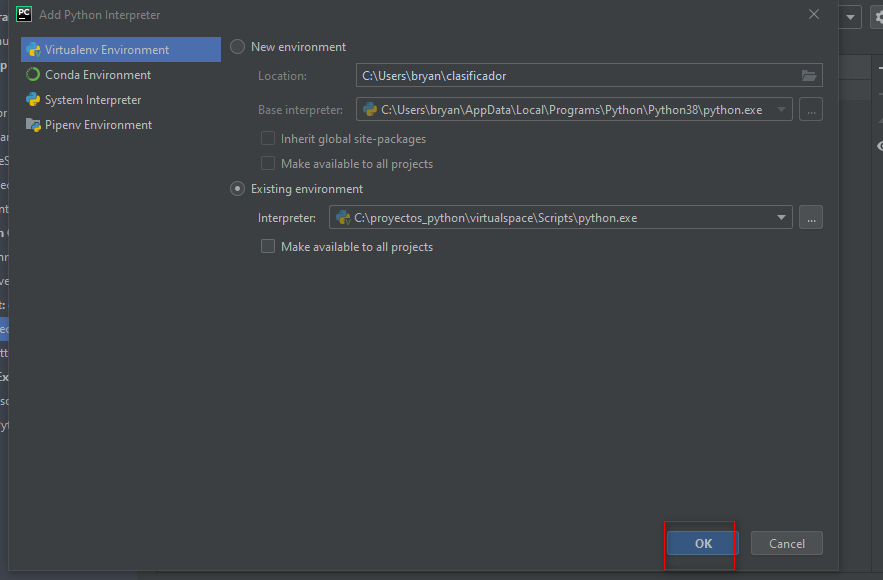
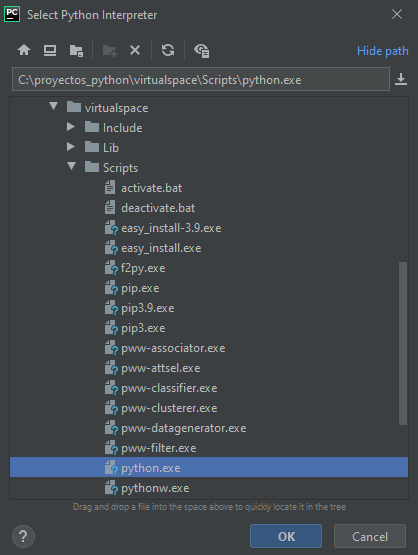
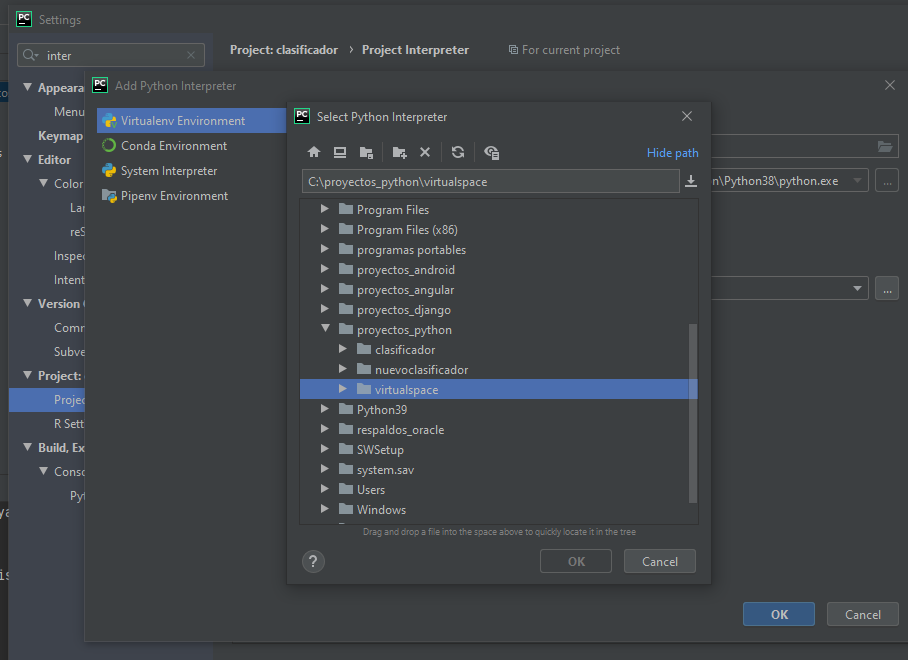
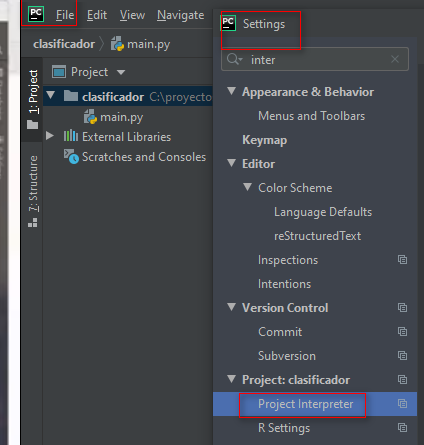




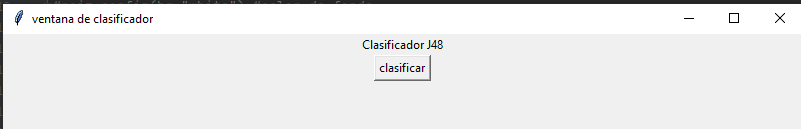
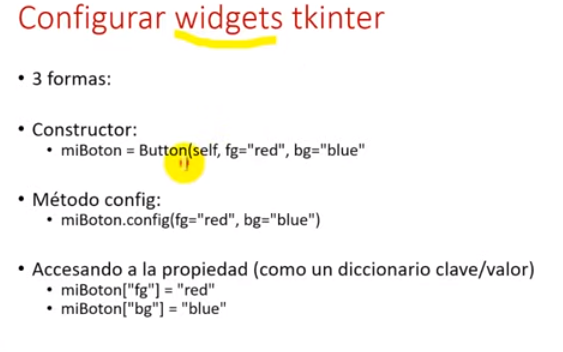
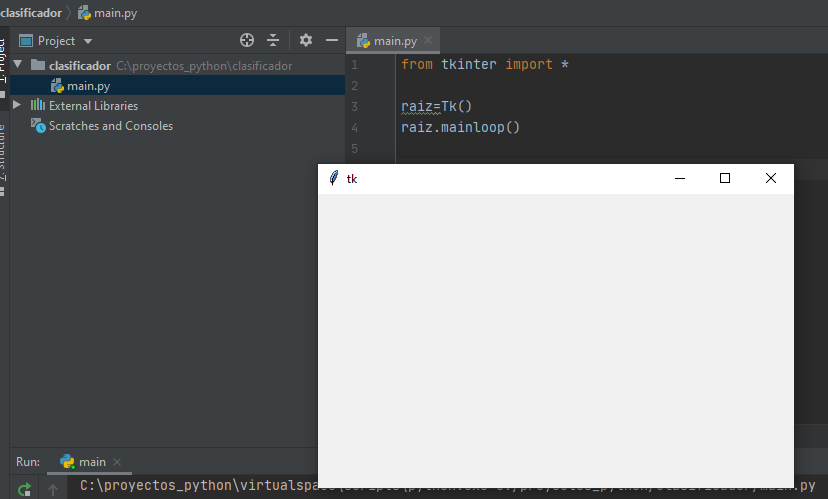
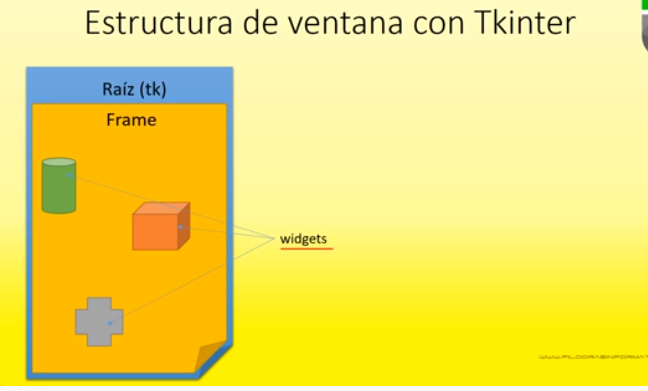
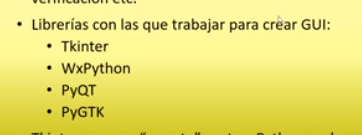








**Crear un formulario con python**



[**http://fracpete.github.io/python-weka-wrapper3/examples.html#location-of-the-datasets**](http://fracpete.github.io/python-weka-wrapper3/examples.html#location-of-the-datasets)

[**http://fracpete.github.io/python-weka-wrapper3/**](http://fracpete.github.io/python-weka-wrapper3/)

[**http://fracpete.github.io/python-weka-wrapper3/index.html**](http://fracpete.github.io/python-weka-wrapper3/index.html)

[**https://opensourcelibs.com/lib/python-weka-wrapper-examples**](https://opensourcelibs.com/lib/python-weka-wrapper-examples)

[**https://www.youtube.com/watch?v=YT72KkkfD3w&t=232s**](https://www.youtube.com/watch?v=YT72KkkfD3w&t=232s)

|  |
| --- |
| from tkinter import \* #importaciones para trabajar con weka from unittest import loader  import weka.core.jvm as jvm import weka.core.converters as conv from weka.classifiers import Evaluation from weka.classifiers import Classifier from weka.core.classes import Random import os from weka.core.converters import Loader #para cargar el dataset from weka.core.dataset import Instance from weka.filters import Filter #para filtrar datos from weka.core.dataset import Attribute, Instance, Instances  # Inicie JVM jvm.start()  #en caso de que este dentro de una carpeta insertamos data\_dir = "../carpeta/carpeta" loader = Loader(classname="weka.core.converters.ArffLoader") data=loader.load\_file("labor.arff") #si el archivo estadentro de una carpeta seria data=loader.load\_file(data\_dir+"labor.arff") data.class\_is\_last()#indicar que la clase respecto a la que debe clasificar es la ultima print(data)   print("\n====== CLASIFICACION J48 ======") j48 = Classifier(classname="weka.classifiers.trees.J48", options=["-U"]) j48.build\_classifier(data) print(j48)  print("====== EVALUACION DEL ARBOL =====") cls = Classifier(classname="weka.classifiers.trees.J48") ev1 = Evaluation(data) #generamos una isntancia de un objeto Evaluacion con os datos de entrenamiento para determinar los antecedentes ev1.crossvalidate\_model(cls, data, 10, Random(1))#validar de forma cruzada el clasificador de datos con una valicacion cruzada de 10 veces #imprimir el resumen de evaluacion print(ev1.summary("\n====== SUMMARY J48 ======", False)) print(ev1.class\_details("====== ACURRACY J48 ======")) #imprimir el resumen de matriz de condusion print(ev1.matrix("====== MATRIZ DE CONFUSION J48 ======")) print("\n====== Cladificar una nueva Instancia ======", False)  num\_att1 = Attribute.create\_numeric("duration") num\_att2 = Attribute.create\_numeric("wage-increase-first-year") num\_att3 = Attribute.create\_numeric("wage-increase-second-year") num\_att4 = Attribute.create\_numeric("wage-increase-third-year") nom\_att5 = Attribute.create\_nominal("cost-of-living-adjustment", ["none", "tcf", "tc"]) num\_att6 = Attribute.create\_numeric("working-hours") nom\_att7 = Attribute.create\_nominal("pension", ["none", "ret\_allw", "empl\_contr"]) num\_att8 = Attribute.create\_numeric("standby-pay") num\_att9 = Attribute.create\_numeric("shift-differential") nom\_att10 = Attribute.create\_nominal("education-allowance", ["yes", "no"]) num\_att11 = Attribute.create\_numeric("statutory-holidays") nom\_att12 = Attribute.create\_nominal("vacation", ["below\_average", "average", "generous"]) nom\_att13 = Attribute.create\_nominal("longterm-disability-assistance", ["yes", "no"]) nom\_att14 = Attribute.create\_nominal("contribution-to-dental-plan", ["none", "half", "full"]) nom\_att15= Attribute.create\_nominal("bereavement-assistance", ["yes", "no"]) nom\_att16 = Attribute.create\_nominal("contribution-to-health-plan", ["none", "half", "full"]) nom\_att17= Attribute.create\_nominal("class", ["bad", "good"])  # create dataset dataset = Instances.create\_instances("labor-neg-data", [num\_att1,num\_att2,num\_att3,num\_att4,nom\_att5,num\_att6,nom\_att7,num\_att8,num\_att9,nom\_att10,num\_att11,nom\_att12,nom\_att13,nom\_att14,nom\_att15,nom\_att16,nom\_att17], 0) # add rows values = [3.0,6.0,6.0,4.0, 1.0,35.0,1.0,2.0,14.0,1.0,9.0,1.0,1.0,1.0,1.0,1.0] inst = Instance.create\_instance(values) data.add\_instance(inst) print(data)     jvm.stop() |

**mmmmmm**

|  |
| --- |
| import re import tkinter as tk from tkinter import StringVar, messagebox from tkinter import ttk from tkinter.ttk import Combobox  #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* #importaciones para trabajar con weka from unittest import loader import weka.core.jvm as jvm import weka.core.converters as conv from weka.classifiers import Evaluation from weka.classifiers import Classifier from weka.core.classes import Random import os from weka.core.converters import Loader #para cargar el dataset from weka.core.dataset import Instance from weka.filters import Filter #para filtrar datos from weka.core.dataset import Attribute, Instance, Instances #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   class FormularioRegistro(tk.Tk):  def \_\_init\_\_(self):  super().\_\_init\_\_()   self.inicializar\_gui()  self.definir\_patrones\_validaciones()   def inicializar\_gui(self):  self.title('Ventana Principal')  self.minsize(400, 500)   lbl\_titulo = tk.Label(self, text="PYTHON | ALGORITMO J48 | DATASET LABOR")  lbl\_titulo.grid(row=0, column=1, pady=10)   frm\_principal = tk.Frame(self, bd=7, relief='groove')  frm\_principal.grid(row=1, column=1, padx=10, pady=10)   # Definicion de campos PRIMERA COLUMNA  # Datos tipo texto  duracion = tk.Label(frm\_principal, text="Duración")  duracion.grid(row=0, column=0, sticky=tk.W)  self.duracion = tk.Entry(frm\_principal, width=20)  self.duracion.grid(row=0, column=1)   incremento1A = tk.Label(frm\_principal, text="Aumento - salario primer año")  incremento1A.grid(row=1, column=0, sticky=tk.W)  self.incremento1A = tk.Entry(frm\_principal, width=20)  self.incremento1A.grid(row=1, column=1)   incremento2A = tk.Label(frm\_principal, text="Aumento - salario segundo año")  incremento2A.grid(row=2, column=0, sticky=tk.W)  self.incremento2A = tk.Entry(frm\_principal, width=20)  self.incremento2A.grid(row=2, column=1)   incremento3A = tk.Label(frm\_principal, text="Aumento - salario tercer año")  incremento3A.grid(row=3, column=0, sticky=tk.W)  self.incremento3A = tk.Entry(frm\_principal, width=20)  self.incremento3A.grid(row=3, column=1)   # Tipo de dato seleccion  ajusteCostoVida = tk.Label(frm\_principal, text="Ajuste por costo de vida")  ajusteCostoVida.grid(row=4, column=0, sticky=tk.W)  self.opcionesACV = ["none", "tfc", "tc"]  self.ajusteCostoVida = Combobox(self, width="10", values=self.opcionesACV, state="readonly")  self.ajusteCostoVida.place(x=191, y=143)   # Dato tipo texto  horasTrabajo = tk.Label(frm\_principal, text="Horas Laborables")  horasTrabajo.grid(row=5, column=0, sticky=tk.W)  self.horasTrabajo = tk.Entry(frm\_principal, width=20)  self.horasTrabajo.grid(row=5, column=1)   # Tipo de dato seleccion  pension = tk.Label(frm\_principal, text="Pensión")  pension.grid(row=6, column=0, sticky=tk.W)  self.opcionesP = ["none", "ret\_allw", "empl\_contr"]  self.pension = Combobox(self, width="10", values=self.opcionesP, state="readonly")  self.pension.place(x=191, y=185)   # Dato tipo texto  pagoEnEspera = tk.Label(frm\_principal, text="Pago en espera")  pagoEnEspera.grid(row=7, column=0, sticky=tk.W)  self.pagoEnEspera = tk.Entry(frm\_principal, width=20)  self.pagoEnEspera.grid(row=7, column=1)   diferencialDelCambio = tk.Label(frm\_principal, text="Diferencial de cambio")  diferencialDelCambio.grid(row=8, column=0, sticky=tk.W)  self.diferencialDelCambio = tk.Entry(frm\_principal, width=20)  self.diferencialDelCambio.grid(row=8, column=1)   # Definicion de campos SEGUNDA COLUMNA  # Tipo de dato seleccion  subsidioEducacion = tk.Label(frm\_principal, text="Subsidio educación")  subsidioEducacion.grid(row=9, column=0, sticky=tk.W)  self.opcionesSE = ["yes", "no"]  self.subsidioEducacion = Combobox(self, width="10", values=self.opcionesSE, state="readonly")  self.subsidioEducacion.place(x=191, y=248)   # Dato tipo texto  diasFeriados = tk.Label(frm\_principal, text="Días feriados")  diasFeriados.grid(row=10, column=0, sticky=tk.W)  self.diasFeriados = tk.Entry(frm\_principal, width=20)  self.diasFeriados.grid(row=10, column=1)   # Tipo de dato seleccion  vacaciones = tk.Label(frm\_principal, text="Vacaciones")  vacaciones.grid(row=11, column=0, sticky=tk.W)  self.opcionesV = ["below\_average", "average", "generous"]  self.vacaciones = Combobox(self, width="10", values=self.opcionesV, state="readonly")  self.vacaciones.place(x=191, y=290)   asistenciaDiscapacidad = tk.Label(frm\_principal, text="Asistencia discapacidad")  asistenciaDiscapacidad.grid(row=12, column=0, sticky=tk.W)  self.opcionesAD = ["yes", "no"]  self.asistenciaDiscapacidad = Combobox(self, width="10", values=self.opcionesAD, state="readonly")  self.asistenciaDiscapacidad.place(x=191, y=311)   contribucionPlanDental = tk.Label(frm\_principal, text="Contribución al plan dental")  contribucionPlanDental.grid(row=13, column=0, sticky=tk.W)  self.opcionesCPD = ["none", "half", "full"]  self.contribucionPlanDental = Combobox(self, width="10", values=self.opcionesCPD, state="readonly")  self.contribucionPlanDental.place(x=191, y=332)   ayudaDuelo = tk.Label(frm\_principal, text="Ayuda para el duelo")  ayudaDuelo.grid(row=14, column=0, sticky=tk.W)  self.opcionesAyudaD = ["yes", "no"]  self.ayudaDuelo = Combobox(self, width="10", values=self.opcionesAyudaD, state="readonly")  self.ayudaDuelo.place(x=191, y=351)   contribucionPlanSalud = tk.Label(frm\_principal, text="Contribución al plan de salud")  contribucionPlanSalud.grid(row=15, column=0, sticky=tk.W)  self.opcionesCPS = ["none", "half", "full"]  self.contribucionPlanSalud = Combobox(self, width="10", values=self.opcionesCPS, state="readonly")  self.contribucionPlanSalud.place(x=191, y=372)   # Botones  btn\_guardar = tk.Button(frm\_principal, text='Guardar', command=self.guardar)  btn\_guardar.grid(row=17, column=2)   btn\_limpiar = tk.Button(frm\_principal, text='Limpiar', command=self.limpiar)  btn\_limpiar.grid(row=17, column=3)   btn\_salir = tk.Button(frm\_principal, text='Salir', command=self.salir)  btn\_salir.grid(row=17, column=4)   # Boton clasificar  btn\_clasificar = tk.Button(frm\_principal, text='Clasificar', command=self.clasificar)  btn\_clasificar.grid(row=19, column=0, sticky=tk.W)   self.btn\_clasificar = tk.Entry(frm\_principal, width=20)  self.btn\_clasificar.grid(row=19, column=1)   # Validaciones  def definir\_patrones\_validaciones(self):  patron\_duracion = r'^[0-9]{1,10}$'  self.regex\_duracion = re.compile(patron\_duracion)   patron\_incremento1A = r'^[0-9]{1,10}$'  self.regex\_incremento1A = re.compile(patron\_incremento1A)   patron\_incremento2A = r'^[0-9]{1,10}$'  self.regex\_incremento2A = re.compile(patron\_incremento2A)   patron\_incremento3A = r'^[0-9]{1,10}$'  self.regex\_incremento3A = re.compile(patron\_incremento3A)   patron\_horasTrabajo = r'^[0-9]{1,10}$'  self.regex\_horasTrabajo = re.compile(patron\_horasTrabajo)   patron\_pagoEnEspera = r'^[0-9]{1,10}$'  self.regex\_pagoEnEspera = re.compile(patron\_pagoEnEspera)   patron\_diferencialDelCambio = r'^[0-9]{1,10}$'  self.regex\_diferencialDelCambio = re.compile(patron\_diferencialDelCambio)   patron\_diasFeriados = r'^[0-9]{1,10}$'  self.regex\_diasFeriados = re.compile(patron\_diasFeriados)   # Funciones de los botones  def guardar(self):   duracion = self.duracion.get().strip()  if re.match(self.regex\_duracion, duracion) is None:  messagebox.showwarning('Mensaje', 'El campo "Duración"debe ser numerico')  return   incremento1A = self.incremento1A.get().strip()  if re.match(self.regex\_incremento1A, incremento1A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario primer año" debe ser numerico')  return   incremento2A = self.incremento2A.get().strip()  if re.match(self.regex\_incremento2A, incremento2A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario segundo año" debe ser numerico')  return   incremento3A = self.incremento3A.get().strip()  if re.match(self.regex\_incremento3A, incremento3A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario tercer año" debe ser numerico')  return   self.ajusteCostoVida.get()   horasTrabajo = self.horasTrabajo.get().strip()  if re.match(self.regex\_horasTrabajo, horasTrabajo) is None:  messagebox.showwarning('Mensaje', 'El campo "Horas Laborables" debe ser numerico')  return   self.pension.get()   pagoEnEspera = self.pagoEnEspera.get().strip()  if re.match(self.regex\_pagoEnEspera, pagoEnEspera) is None:  messagebox.showwarning('Mensaje', 'El campo "Pago en espera" debe ser numerico')  return   diferencialDelCambio = self.diferencialDelCambio.get().strip()  if re.match(self.regex\_diferencialDelCambio, diferencialDelCambio) is None:  messagebox.showwarning('Mensaje', 'El "Diferencial del Cambio" campo debe ser numerico')  return   self.subsidioEducacion.get()   diasFeriados = self.diasFeriados.get().strip()  if re.match(self.regex\_diasFeriados, diasFeriados) is None:  messagebox.showwarning('Mensaje', 'El campo "Dias Feriados" debe ser numerico')  return   self.vacaciones.get()   self.asistenciaDiscapacidad.get()   self.contribucionPlanDental.get()   self.ayudaDuelo.get()   self.contribucionPlanSalud.get   messagebox.showinfo('Mensaje', 'Los datos se guardaron')   # Transformando los datos a float  duracion = float(duracion)  incremento1A = float(incremento1A)  incremento2A = float(incremento2A)  incremento3A = float(incremento3A)   horasTrabajo = float(horasTrabajo)  pagoEnEspera = float(pagoEnEspera)  diferencialDelCambio = float(diferencialDelCambio)  diasFeriados = float(diasFeriados)   print(duracion, "\t", incremento1A, "\t", incremento2A, "\t", incremento3A, "\t",  self.ajusteCostoVida.get(), "\t", horasTrabajo, "\t", self.pension.get(), "\t", pagoEnEspera, "\t\n",  diferencialDelCambio, "\t", self.subsidioEducacion.get(), "\t", diasFeriados, "\t", self.vacaciones.get(),  "\t\n",  self.asistenciaDiscapacidad.get(), "\t", self.contribucionPlanDental.get(), "\t", self.ayudaDuelo.get(),  "\t", self.contribucionPlanSalud.get)   # Funcion limpiar   def limpiar(self):  self.duracion.delete(0, 'end')  self.incremento1A.delete(0, 'end')  self.incremento2A.delete(0, 'end')  self.incremento3A.delete(0, 'end')  self.ajusteCostoVida.current(0)  self.horasTrabajo.delete(0, 'end')  self.pension.current(0)  self.pagoEnEspera.delete(0, 'end')  self.diferencialDelCambio.delete(0, 'end')  self.subsidioEducacion.current(0)  self.diasFeriados.delete(0, 'end')  self.vacaciones.current(0)  self.asistenciaDiscapacidad.current(0)  self.contribucionPlanDental.current(0)  self.ayudaDuelo.current(0)  self.contribucionPlanSalud.current(0)   def salir(self):  self.destroy()   def clasificar(self):  *""""  duracion = self.duracion.get().strip()  incremento1A = self.incremento1A.get().strip()  incremento2A = self.incremento2A.get().strip()  incremento3A = self.incremento3A.get().strip()  self.ajusteCostoVida.get()  self.pension.get()  horasTrabajo = self.horasTrabajo.get().strip()  pagoEnEspera = self.pagoEnEspera.get().strip()  diferencialDelCambio = self.diferencialDelCambio.get().strip()  self.subsidioEducacion.get()  self.vacaciones.get()  self.asistenciaDiscapacidad.get()  self.contribucionPlanDental.get()  self.ayudaDuelo.get()  self.contribucionPlanSalud.get   if re.match(self.regex\_duracion, duracion) is None:  messagebox.showwarning('Mensaje', 'El campo "Duración"debe ser numerico')  return  else:   if re.match(self.regex\_incremento1A, incremento1A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario primer año" debe ser numerico')  return   else:   if re.match(self.regex\_incremento2A, incremento2A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario segundo año" debe ser numerico')  return   else:   if re.match(self.regex\_incremento3A, incremento3A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario tercer año" debe ser numerico')  return   else:   if re.match(self.regex\_horasTrabajo, horasTrabajo) is None:  messagebox.showwarning('Mensaje', 'El campo "Horas Laborables" debe ser numerico')  return  else:   if re.match(self.regex\_pagoEnEspera, pagoEnEspera) is None:  messagebox.showwarning('Mensaje', 'El campo "Pago en espera" debe ser numerico')  return  else:   if re.match(self.regex\_diferencialDelCambio, diferencialDelCambio) is None:  messagebox.showwarning('Mensaje', 'El "Diferencial del Cambio" campo debe ser numerico')  return  else:   diasFeriados = self.diasFeriados.get().strip()  if re.match(self.regex\_diasFeriados, diasFeriados) is None:  messagebox.showwarning('Mensaje', 'El campo "Dias Feriados" debe ser numerico')  return    messagebox.showinfo('Mensaje', 'Los datos se guardaron')  # Transformando los datos a float  duracion = float(duracion)  incremento1A = float(incremento1A)  incremento2A = float(incremento2A)  incremento3A = float(incremento3A)  horasTrabajo = float(horasTrabajo)  pagoEnEspera = float(pagoEnEspera)  diferencialDelCambio = float(diferencialDelCambio)  diasFeriados = float(diasFeriados)   print(duracion, "\t", incremento1A, "\t", incremento2A, "\t", incremento3A, "\t",  self.ajusteCostoVida.get(), "\t", horasTrabajo, "\t", self.pension.get(), "\t", pagoEnEspera, "\t\n",  diferencialDelCambio, "\t", self.subsidioEducacion.get(), "\t", diasFeriados, "\t", self.vacaciones.get(),  "\t\n",  self.asistenciaDiscapacidad.get(), "\t", self.contribucionPlanDental.get(), "\t", self.ayudaDuelo.get(),  "\t", self.contribucionPlanSalud.get)  """* #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*  # Inicie JVM  jvm.start()  # en caso de que este dentro de una carpeta insertamos data\_dir = "../carpeta/carpeta"  loader = Loader(classname="weka.core.converters.ArffLoader")  data = loader.load\_file("labor.arff") # si el archivo estadentro de una carpeta seria data=loader.load\_file(data\_dir+"labor.arff")  data.class\_is\_last() # indicar que la clase respecto a la que debe clasificar es la ultima  print(data)   print("\n====== CLASIFICACION J48 ======")  #j48 = Classifier(classname="weka.classifiers.trees.J48", options=["-U"])  j48 = Classifier(classname="weka.classifiers.trees.J48", options=["-C", "0.3"])  j48.build\_classifier(data)  print(j48)   print("====== EVALUACION DEL ARBOL =====")  cls = Classifier(classname="weka.classifiers.trees.J48")  ev1 = Evaluation(data) # generamos una isntancia de un objeto Evaluacion con os datos de entrenamiento para determinar los antecedentes  ev1.crossvalidate\_model(cls, data, 10, Random(1)) # validar de forma cruzada el clasificador de datos con una valicacion cruzada de 10 veces  # imprimir el resumen de evaluacion  print(ev1.summary("\n====== SUMMARY J48 ======", False))  print(ev1.class\_details("====== ACURRACY J48 ======"))  # imprimir el resumen de matriz de condusion  print(ev1.matrix("====== MATRIZ DE CONFUSION J48 ======"))   #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  print("\n====== Cladificar una nueva Instancia ======", False)  """  #si Deseo crear un dataset manualmente  num\_att1 = Attribute.create\_numeric("duration")  num\_att2 = Attribute.create\_numeric("wage-increase-first-year")  num\_att3 = Attribute.create\_numeric("wage-increase-second-year")  num\_att4 = Attribute.create\_numeric("wage-increase-third-year")  nom\_att5 = Attribute.create\_nominal("cost-of-living-adjustment", ["none", "tcf", "tc"])  num\_att6 = Attribute.create\_numeric("working-hours")  nom\_att7 = Attribute.create\_nominal("pension", ["none", "ret\_allw", "empl\_contr"])  num\_att8 = Attribute.create\_numeric("standby-pay")  num\_att9 = Attribute.create\_numeric("shift-differential")  nom\_att10 = Attribute.create\_nominal("education-allowance", ["yes", "no"])  num\_att11 = Attribute.create\_numeric("statutory-holidays")  nom\_att12 = Attribute.create\_nominal("vacation", ["below\_average", "average", "generous"])  nom\_att13 = Attribute.create\_nominal("longterm-disability-assistance", ["yes", "no"])  nom\_att14 = Attribute.create\_nominal("contribution-to-dental-plan", ["none", "half", "full"])  nom\_att15 = Attribute.create\_nominal("bereavement-assistance", ["yes", "no"])  nom\_att16 = Attribute.create\_nominal("contribution-to-health-plan", ["none", "half", "full"])  nom\_att17 = Attribute.create\_nominal("class", ["bad", "good"])  # create dataset  dataset = Instances.create\_instances("labor-neg-data",[num\_att1, num\_att2, num\_att3, num\_att4, nom\_att5, num\_att6, nom\_att7,  num\_att8, num\_att9, nom\_att10, num\_att11, nom\_att12, nom\_att13, nom\_att14,  nom\_att15, nom\_att16, nom\_att17], 0)  """  # add rows  #values = [3.0, 6.0, 6.0, 4.0, 1.0, 35.0, 1.0, 2.0, 14.0, 1.0, 9.0, 1.0, 1.0, 1.0, 1.0, 1.0] #clasifica como Bad  #values = [3.0, 6.0, 6.0, 4.0, 0.0, 35.0, 0.0, 2.0, 14.0, 0.0, 9.0, 2.0, 0.0, 2.0, 0.0, 2.0] #clasifica como BAd  values = [9.0, 9.0, 9.0, 9.0, 0.0, 50.0, 0.0, 2.0, 14.0, 0.0, 14.0, 2.0, 0.0, 2.0, 0.0, 2.0] #clasifica como Good  inst = Instance.create\_instance(values) #creamos la instancia con los datos values  #print(inst)#si quiero ver la instancia qie se esta ingresando  data.add\_instance(inst) #inserto la instancia en los datos  #print(data) #si quiero ver los datos para comprovar la inserccion   for index, inst in enumerate(data):  pred = j48.classify\_instance(inst)  dist = j48.distribution\_for\_instance(inst)  #print(str(index + 1) + ": label index=" + str(pred) + ", class distribution=" + str(dist))   print(data.class\_attribute.value(pred)) #imprimo el resultado de la clasificacion   jvm.stop() ##cerramos el bridgue   #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*  def main():  app = FormularioRegistro()  app.mainloop()   if \_\_name\_\_ == '\_\_main\_\_':  main() |

|  |
| --- |
| **Ya cladifica finalmente** |

|  |
| --- |
| import re import tkinter as tk from tkinter import StringVar, messagebox from tkinter import ttk from tkinter.ttk import Combobox  #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* #importaciones para trabajar con weka from unittest import loader import weka.core.jvm as jvm import weka.core.converters as conv from weka.classifiers import Evaluation from weka.classifiers import Classifier from weka.core.classes import Random import os from weka.core.converters import Loader #para cargar el dataset from weka.core.dataset import Instance from weka.filters import Filter #para filtrar datos from weka.core.dataset import Attribute, Instance, Instances #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   class FormularioRegistro(tk.Tk):  def \_\_init\_\_(self):  super().\_\_init\_\_()   self.inicializar\_gui()  self.definir\_patrones\_validaciones()   def inicializar\_gui(self):  self.title('Ventana Principal')  self.minsize(400, 500)   lbl\_titulo = tk.Label(self, text="PYTHON | ALGORITMO J48 | DATASET LABOR")  lbl\_titulo.grid(row=0, column=1, pady=10)   frm\_principal = tk.Frame(self, bd=7, relief='groove')  frm\_principal.grid(row=1, column=1, padx=10, pady=10)   # Definicion de campos PRIMERA COLUMNA  # Datos tipo texto  duracion = tk.Label(frm\_principal, text="Duración")  duracion.grid(row=0, column=0, sticky=tk.W)  self.duracion = tk.Entry(frm\_principal, width=20)  self.duracion.grid(row=0, column=1)   incremento1A = tk.Label(frm\_principal, text="Aumento - salario primer año")  incremento1A.grid(row=1, column=0, sticky=tk.W)  self.incremento1A = tk.Entry(frm\_principal, width=20)  self.incremento1A.grid(row=1, column=1)   incremento2A = tk.Label(frm\_principal, text="Aumento - salario segundo año")  incremento2A.grid(row=2, column=0, sticky=tk.W)  self.incremento2A = tk.Entry(frm\_principal, width=20)  self.incremento2A.grid(row=2, column=1)   incremento3A = tk.Label(frm\_principal, text="Aumento - salario tercer año")  incremento3A.grid(row=3, column=0, sticky=tk.W)  self.incremento3A = tk.Entry(frm\_principal, width=20)  self.incremento3A.grid(row=3, column=1)   # Tipo de dato seleccion  ajusteCostoVida = tk.Label(frm\_principal, text="Ajuste por costo de vida")  ajusteCostoVida.grid(row=4, column=0, sticky=tk.W)  self.opcionesACV = ["none", "tfc", "tc"]  self.ajusteCostoVida = Combobox(self, width="10", values=self.opcionesACV, state="readonly")  self.ajusteCostoVida.place(x=191, y=143)   # Dato tipo texto  horasTrabajo = tk.Label(frm\_principal, text="Horas Laborables")  horasTrabajo.grid(row=5, column=0, sticky=tk.W)  self.horasTrabajo = tk.Entry(frm\_principal, width=20)  self.horasTrabajo.grid(row=5, column=1)   # Tipo de dato seleccion  pension = tk.Label(frm\_principal, text="Pensión")  pension.grid(row=6, column=0, sticky=tk.W)  self.opcionesP = ["none", "ret\_allw", "empl\_contr"]  self.pension = Combobox(self, width="10", values=self.opcionesP, state="readonly")  self.pension.place(x=191, y=185)   # Dato tipo texto  pagoEnEspera = tk.Label(frm\_principal, text="Pago en espera")  pagoEnEspera.grid(row=7, column=0, sticky=tk.W)  self.pagoEnEspera = tk.Entry(frm\_principal, width=20)  self.pagoEnEspera.grid(row=7, column=1)   diferencialDelCambio = tk.Label(frm\_principal, text="Diferencial de cambio")  diferencialDelCambio.grid(row=8, column=0, sticky=tk.W)  self.diferencialDelCambio = tk.Entry(frm\_principal, width=20)  self.diferencialDelCambio.grid(row=8, column=1)   # Definicion de campos SEGUNDA COLUMNA  # Tipo de dato seleccion  subsidioEducacion = tk.Label(frm\_principal, text="Subsidio educación")  subsidioEducacion.grid(row=9, column=0, sticky=tk.W)  self.opcionesSE = ["yes", "no"]  self.subsidioEducacion = Combobox(self, width="10", values=self.opcionesSE, state="readonly")  self.subsidioEducacion.place(x=191, y=248)   # Dato tipo texto  diasFeriados = tk.Label(frm\_principal, text="Días feriados")  diasFeriados.grid(row=10, column=0, sticky=tk.W)  self.diasFeriados = tk.Entry(frm\_principal, width=20)  self.diasFeriados.grid(row=10, column=1)   # Tipo de dato seleccion  vacaciones = tk.Label(frm\_principal, text="Vacaciones")  vacaciones.grid(row=11, column=0, sticky=tk.W)  self.opcionesV = ["below\_average", "average", "generous"]  self.vacaciones = Combobox(self, width="10", values=self.opcionesV, state="readonly")  self.vacaciones.place(x=191, y=290)   asistenciaDiscapacidad = tk.Label(frm\_principal, text="Asistencia discapacidad")  asistenciaDiscapacidad.grid(row=12, column=0, sticky=tk.W)  self.opcionesAD = ["yes", "no"]  self.asistenciaDiscapacidad = Combobox(self, width="10", values=self.opcionesAD, state="readonly")  self.asistenciaDiscapacidad.place(x=191, y=311)   contribucionPlanDental = tk.Label(frm\_principal, text="Contribución al plan dental")  contribucionPlanDental.grid(row=13, column=0, sticky=tk.W)  self.opcionesCPD = ["none", "half", "full"]  self.contribucionPlanDental = Combobox(self, width="10", values=self.opcionesCPD, state="readonly")  self.contribucionPlanDental.place(x=191, y=332)   ayudaDuelo = tk.Label(frm\_principal, text="Ayuda para el duelo")  ayudaDuelo.grid(row=14, column=0, sticky=tk.W)  self.opcionesAyudaD = ["yes", "no"]  self.ayudaDuelo = Combobox(self, width="10", values=self.opcionesAyudaD, state="readonly")  self.ayudaDuelo.place(x=191, y=351)   contribucionPlanSalud = tk.Label(frm\_principal, text="Contribución al plan de salud")  contribucionPlanSalud.grid(row=15, column=0, sticky=tk.W)  self.opcionesCPS = ["none", "half", "full"]  self.contribucionPlanSalud = Combobox(self, width="10", values=self.opcionesCPS, state="readonly")  self.contribucionPlanSalud.place(x=191, y=372)   # Botones  btn\_guardar = tk.Button(frm\_principal, text='Guardar', command=self.guardar)  btn\_guardar.grid(row=17, column=2)   btn\_limpiar = tk.Button(frm\_principal, text='Limpiar', command=self.limpiar)  btn\_limpiar.grid(row=17, column=3)   btn\_salir = tk.Button(frm\_principal, text='Salir', command=self.salir)  btn\_salir.grid(row=17, column=4)   # Boton clasificar  btn\_clasificar = tk.Button(frm\_principal, text='Clasificar', command=self.clasificar)  btn\_clasificar.grid(row=19, column=0, sticky=tk.W)   self.btn\_clasificar = tk.Entry(frm\_principal, width=20)  self.btn\_clasificar.grid(row=19, column=1)     # Validaciones  def definir\_patrones\_validaciones(self):  patron\_duracion = r'^[0-9]{1,10}$'  self.regex\_duracion = re.compile(patron\_duracion)   patron\_incremento1A = r'^[0-9]{1,10}$'  self.regex\_incremento1A = re.compile(patron\_incremento1A)   patron\_incremento2A = r'^[0-9]{1,10}$'  self.regex\_incremento2A = re.compile(patron\_incremento2A)   patron\_incremento3A = r'^[0-9]{1,10}$'  self.regex\_incremento3A = re.compile(patron\_incremento3A)   patron\_horasTrabajo = r'^[0-9]{1,10}$'  self.regex\_horasTrabajo = re.compile(patron\_horasTrabajo)   patron\_pagoEnEspera = r'^[0-9]{1,10}$'  self.regex\_pagoEnEspera = re.compile(patron\_pagoEnEspera)   patron\_diferencialDelCambio = r'^[0-9]{1,10}$'  self.regex\_diferencialDelCambio = re.compile(patron\_diferencialDelCambio)   patron\_diasFeriados = r'^[0-9]{1,10}$'  self.regex\_diasFeriados = re.compile(patron\_diasFeriados)   # Funciones de los botones  def guardar(self):   duracion = self.duracion.get().strip()  if re.match(self.regex\_duracion, duracion) is None:  messagebox.showwarning('Mensaje', 'El campo "Duración"debe ser numerico')  return   incremento1A = self.incremento1A.get().strip()  if re.match(self.regex\_incremento1A, incremento1A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario primer año" debe ser numerico')  return   incremento2A = self.incremento2A.get().strip()  if re.match(self.regex\_incremento2A, incremento2A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario segundo año" debe ser numerico')  return   incremento3A = self.incremento3A.get().strip()  if re.match(self.regex\_incremento3A, incremento3A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario tercer año" debe ser numerico')  return   self.ajusteCostoVida.get()   horasTrabajo = self.horasTrabajo.get().strip()  if re.match(self.regex\_horasTrabajo, horasTrabajo) is None:  messagebox.showwarning('Mensaje', 'El campo "Horas Laborables" debe ser numerico')  return   self.pension.get()   pagoEnEspera = self.pagoEnEspera.get().strip()  if re.match(self.regex\_pagoEnEspera, pagoEnEspera) is None:  messagebox.showwarning('Mensaje', 'El campo "Pago en espera" debe ser numerico')  return   diferencialDelCambio = self.diferencialDelCambio.get().strip()  if re.match(self.regex\_diferencialDelCambio, diferencialDelCambio) is None:  messagebox.showwarning('Mensaje', 'El "Diferencial del Cambio" campo debe ser numerico')  return   self.subsidioEducacion.get()   diasFeriados = self.diasFeriados.get().strip()  if re.match(self.regex\_diasFeriados, diasFeriados) is None:  messagebox.showwarning('Mensaje', 'El campo "Dias Feriados" debe ser numerico')  return   self.vacaciones.get()   self.asistenciaDiscapacidad.get()   self.contribucionPlanDental.get()   self.ayudaDuelo.get()   self.contribucionPlanSalud.get   messagebox.showinfo('Mensaje', 'Los datos se guardaron')   # Transformando los datos a float  duracion = float(duracion)  incremento1A = float(incremento1A)  incremento2A = float(incremento2A)  incremento3A = float(incremento3A)   horasTrabajo = float(horasTrabajo)  pagoEnEspera = float(pagoEnEspera)  diferencialDelCambio = float(diferencialDelCambio)  diasFeriados = float(diasFeriados)   print(duracion, "\t", incremento1A, "\t", incremento2A, "\t", incremento3A, "\t",  self.ajusteCostoVida.get(), "\t", horasTrabajo, "\t", self.pension.get(), "\t", pagoEnEspera, "\t\n",  diferencialDelCambio, "\t", self.subsidioEducacion.get(), "\t", diasFeriados, "\t", self.vacaciones.get(),  "\t\n",  self.asistenciaDiscapacidad.get(), "\t", self.contribucionPlanDental.get(), "\t", self.ayudaDuelo.get(),  "\t", self.contribucionPlanSalud.get)   # Funcion limpiar   def limpiar(self):  self.duracion.delete(0, 'end')  self.incremento1A.delete(0, 'end')  self.incremento2A.delete(0, 'end')  self.incremento3A.delete(0, 'end')  self.ajusteCostoVida.current(0)  self.horasTrabajo.delete(0, 'end')  self.pension.current(0)  self.pagoEnEspera.delete(0, 'end')  self.diferencialDelCambio.delete(0, 'end')  self.subsidioEducacion.current(0)  self.diasFeriados.delete(0, 'end')  self.vacaciones.current(0)  self.asistenciaDiscapacidad.current(0)  self.contribucionPlanDental.current(0)  self.ayudaDuelo.current(0)  self.contribucionPlanSalud.current(0)   def salir(self):  self.destroy()   def clasificar(self):  duracion = self.duracion.get().strip()  incremento1A = self.incremento1A.get().strip()  incremento2A = self.incremento2A.get().strip()  incremento3A = self.incremento3A.get().strip()  self.ajusteCostoVida.get()  self.pension.get()  horasTrabajo = self.horasTrabajo.get().strip()  pagoEnEspera = self.pagoEnEspera.get().strip()  diferencialDelCambio = self.diferencialDelCambio.get().strip()  self.subsidioEducacion.get()  self.vacaciones.get()  self.asistenciaDiscapacidad.get()  self.contribucionPlanDental.get()  self.ayudaDuelo.get()  self.contribucionPlanSalud.get   if re.match(self.regex\_duracion, duracion) is None:  messagebox.showwarning('Mensaje', 'El campo "Duración"debe ser numerico')  return  else:   if re.match(self.regex\_incremento1A, incremento1A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario primer año" debe ser numerico')  return   else:   if re.match(self.regex\_incremento2A, incremento2A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario segundo año" debe ser numerico')  return   else:   if re.match(self.regex\_incremento3A, incremento3A) is None:  messagebox.showwarning('Mensaje', 'El campo "Aumento - salario tercer año" debe ser numerico')  return   else:   if re.match(self.regex\_horasTrabajo, horasTrabajo) is None:  messagebox.showwarning('Mensaje', 'El campo "Horas Laborables" debe ser numerico')  return  else:   if re.match(self.regex\_pagoEnEspera, pagoEnEspera) is None:  messagebox.showwarning('Mensaje', 'El campo "Pago en espera" debe ser numerico')  return  else:   if re.match(self.regex\_diferencialDelCambio, diferencialDelCambio) is None:  messagebox.showwarning('Mensaje', 'El "Diferencial del Cambio" campo debe ser numerico')  return  else:   diasFeriados = self.diasFeriados.get().strip()  if re.match(self.regex\_diasFeriados, diasFeriados) is None:  messagebox.showwarning('Mensaje', 'El campo "Dias Feriados" debe ser numerico')  return    messagebox.showinfo('Mensaje', 'Los datos se guardaron')  # Transformando los datos a float  duracion = float(duracion)  incremento1A = float(incremento1A)  incremento2A = float(incremento2A)  incremento3A = float(incremento3A)  horasTrabajo = float(horasTrabajo)  pagoEnEspera = float(pagoEnEspera)  diferencialDelCambio = float(diferencialDelCambio)  diasFeriados = float(diasFeriados)   #transformar las opciones de texto  ajusteCostoVidaNumber =0.0  pensionNumber = 0.0  subcidioEducacionNumber = 0.0  vacacionesNumber = 0.0  asistenciaDiscapacidadNumber = 0.0  contribucionPlanDEntalNumber=0.0  ayudaDueloNumber=0.0  contribucionPlanSalud =0.0    #print(duracion, "\t", incremento1A, "\t", incremento2A, "\t", incremento3A, "\t",  # self.ajusteCostoVida.get(), "\t", horasTrabajo, "\t", self.pension.get(), "\t", pagoEnEspera, "\t\n",  # diferencialDelCambio, "\t", self.subsidioEducacion.get(), "\t", diasFeriados, "\t", self.vacaciones.get(),  # "\t\n",  # self.asistenciaDiscapacidad.get(), "\t", self.contribucionPlanDental.get(), "\t", self.ayudaDuelo.get(),  # "\t", self.contribucionPlanSalud.get())    if self.ajusteCostoVida.get() == "none":  ajusteCostoVidaNumber = 0.0  elif self.ajusteCostoVida.get() == "tfc":  ajusteCostoVidaNumber = 1.0  elif self.ajusteCostoVida.get() == "tc":  ajusteCostoVidaNumber = 2.0   if self.pension.get() == "none":  pensionNumber = 0.0  elif self.pension.get() == "ret\_allw":  pensionNumber = 1.0  elif self.pension.get() == "empl\_contr":  pensionNumber = 2.0   if self.subsidioEducacion.get() == "yes":  subcidioEducacionNumber = 0.0  elif self.subsidioEducacion.get() == "no":  subcidioEducacionNumber= 1.0   if self.vacaciones.get() == "below\_average":  vacacionesNumber = 0.0  elif self.vacaciones.get() == "average":  vacacionesNumber = 1.0  elif self.vacaciones.get() == "generous":  vacacionesNumber = 2.0   if self.asistenciaDiscapacidad.get() == "yes":  asistenciaDiscapacidadNumber = 0.0  elif self.asistenciaDiscapacidad.get() == "no":  asistenciaDiscapacidadNumber= 1.0   if self.contribucionPlanDental.get() == "none":  contribucionPlanDEntalNumber = 0.0  elif self.contribucionPlanDental.get() == "half":  contribucionPlanDEntalNumber = 1.0  elif self.contribucionPlanDental.get() == "full":  contribucionPlanDEntalNumber = 2.0   if self.ayudaDuelo.get() == "yes":  ayudaDueloNumber = 0.0  elif self.ayudaDuelo.get() == "no":  ayudaDueloNumber= 1.0   if self.contribucionPlanSalud.get() == "none":  contribucionPlanSalud = 0.0  elif self.contribucionPlanSalud.get() == "half":  contribucionPlanSalud = 1.0  elif self.contribucionPlanSalud.get() == "full":  contribucionPlanSalud = 2.0   print(ajusteCostoVidaNumber)  print(pensionNumber)  print(subcidioEducacionNumber)  print(vacacionesNumber)  print(asistenciaDiscapacidadNumber)  print(contribucionPlanDEntalNumber)  print(ayudaDueloNumber)  print(contribucionPlanSalud)    #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*  # Inicie JVM  jvm.start()  # en caso de que este dentro de una carpeta insertamos data\_dir = "../carpeta/carpeta"  loader = Loader(classname="weka.core.converters.ArffLoader")  data = loader.load\_file("labor.arff") # si el archivo estadentro de una carpeta seria data=loader.load\_file(data\_dir+"labor.arff")  data.class\_is\_last() # indicar que la clase respecto a la que debe clasificar es la ultima  print(data)   print("\n====== CLASIFICACION J48 ======")  j48 = Classifier(classname="weka.classifiers.trees.J48", options=["-U"])  #j48 = Classifier(classname="weka.classifiers.trees.J48", options=["-C", "0.3"])  j48.build\_classifier(data)  print(j48)   print("====== EVALUACION DEL ARBOL =====")  cls = Classifier(classname="weka.classifiers.trees.J48")  ev1 = Evaluation(data) # generamos una isntancia de un objeto Evaluacion con os datos de entrenamiento para determinar los antecedentes  ev1.crossvalidate\_model(cls, data, 10, Random(1)) # validar de forma cruzada el clasificador de datos con una valicacion cruzada de 10 veces  # imprimir el resumen de evaluacion  print(ev1.summary("\n====== SUMMARY J48 ======", False))  print(ev1.class\_details("====== ACURRACY J48 ======"))  # imprimir el resumen de matriz de condusion  print(ev1.matrix("====== MATRIZ DE CONFUSION J48 ======"))   #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  print("\n====== Cladificar una nueva Instancia ======", False)   # add rows  values = [duracion, incremento1A, incremento2A, incremento3A, ajusteCostoVidaNumber, horasTrabajo, pensionNumber, pagoEnEspera, diferencialDelCambio, subcidioEducacionNumber, diasFeriados, vacacionesNumber, asistenciaDiscapacidadNumber, contribucionPlanDEntalNumber, ayudaDueloNumber,contribucionPlanSalud]  #values = [3.0, 6.0, 6.0, 4.0, 1.0, 35.0, 1.0, 2.0, 14.0, 1.0, 9.0, 1.0, 1.0, 1.0, 1.0, 1.0] #clasifica como Bad  #values = [3.0, 6.0, 6.0, 4.0, 0.0, 35.0, 0.0, 2.0, 14.0, 0.0, 9.0, 2.0, 0.0, 2.0, 0.0, 2.0] #clasifica como Good  #values = [9.0, 9.0, 9.0, 9.0, 0.0, 50.0, 0.0, 2.0, 14.0, 0.0, 14.0, 2.0, 0.0, 2.0, 0.0, 2.0] #clasifica como Good  inst = Instance.create\_instance(values) #creamos la instancia con los datos values  #print(inst)#si quiero ver la instancia qie se esta ingresando  data.add\_instance(inst) #inserto la instancia en los datos  #print(data) #si quiero ver los datos para comprovar la inserccion   for index, inst in enumerate(data):  pred = j48.classify\_instance(inst)  dist = j48.distribution\_for\_instance(inst)  #print(str(index + 1) + ": label index=" + str(pred) + ", class distribution=" + str(dist))   print(data.class\_attribute.value(pred)) #imprimo el resultado de la clasificacion  resultadoClasificacion = data.class\_attribute.value(pred)  self.btn\_clasificar.insert(0,resultadoClasificacion)   jvm.stop() ##cerramos el bridgue   #\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* weka \*\*\*\*\*\*  def main():  app = FormularioRegistro()  app.mainloop()   if \_\_name\_\_ == '\_\_main\_\_':  main() |