Artículo
Ariel Arias · 8 nov, 2021 Lectura de 8 min

Utilizar modelo de IRISFHIRML (NoShow) teniendo origen de datos: TrakCare 2017

MODELO RESUMIDO

Siguiendo el ejemplo disponible en GitHub FHIR IntegratedML vamos usar el modelo resumido para entrenar el modelo de predicción de "No Asistirá" (NotShow).

Haremos un match entre los campos de la tabla PackageSample.NoShowMLRow y los campos de TrakCare. El punto de partida en TrakCare será la tabla SQLUser.RBAppointment.

CAMPO	TRAKCARE	OBSERVACIONES
Gender	APPTPAPMIDR->PAPMISexDR->CTSEXGender	Se espera valores "F" y "M"
ScheduledWeekDay	{fn DAYOFWEEK(APPTDateComp) }	Días de la semana Domingo (1) a Sábado (7)
ScheduledWeek	{fn WEEK(APPTDateComp)}	El número de la semana (1 a 52)
WaitingDays	DATEDIFF('dd',APPTBookedDate, APPTDateComp)	Días entre el agendamiento y la fecha del compromiso)
Age	DATEDIFF('YY',APPTPAPMIDR->PAPMIPAPERDR->PAPERDOB,APPTDateComp)	Edad del paciente (en años) al día de la cita.
Hypertension		CIE10: (I10-I15) Enfermedades hipertensivas
Diabetes		CIE10: (E10-E14) Diabetes mellitus
Alcoholism		CIE10: (F10) Trastornos mentales y de comportamiento debidos al consumo de alcohol
Handicap		CIE10: (Z73.6) Problemas relacionados con la limitación de las actividades debido a discapacidad
SMSreceived	APPTConfirmation	Si no tenemos sistema de envío de mensajes, usaremos el campo "Confirmado"
Noshow	APPTStatus	Usaremos el estado de cita para saber si el paciente fue atendido o se presentó al menos al compromiso

Preparamos la consulta para traer los datos de las citas

Para traer los datos desde TrakCare, vamos a preparar la consulta, y teniendo en cuenta los campos que se requieren, la query quedaría así:

```
Select
APPT_PAPMI_DR->PAPMI_PAPER_DR PatientID,
APPT_RowId ApptID,
APPT_DateComp ApptDate,
APPT_PAPMI_DR->PAPMI_Sex_DR->CTSEX_Gender Gender,
{fn DAYOFWEEK(APPT_DateComp)} ScheduledWeekDay,
{fn WEEK(APPT_DateComp)} ScheduledWeek,
```

```
DATEDIFF('dd',APPT_BookedDate, APPT_DateComp) WaitingDays,
DATEDIFF('YY',APPT_PAPMI_DR->PAPMI_PAPER_DR->PAPER_DOB,APPT_DateComp) Age,
CASE APPT_Confirmation
WHEN 'Y' THEN 1
ELSE 0
END SMSreceived,
CASE APPT_Status
WHEN 'P' THEN 1
WHEN 'A' THEN 0
WHEN 'X' THEN 0
WHEN 'I' THEN 0
WHEN 'N' THEN 1
WHEN 'T' THEN 0
WHEN 'H' THEN 0
WHEN 'C' THEN 0
WHEN 'S' THEN 0
WHEN 'D' THEN 0
ELSE APPT_Status
END Noshow
from RB_Appointment
```

Se puede modificar para traer todos los registros o filtrar por ejemplo el año en curso (dejar al menos fuera las citas futuras, pues estas servirán posteriormente, una vez entrenado el modelo para predecir el comportamiento de citas futuras. Se puede igualmente filtrar por establecimiento (APPTASParRef->ASRESParRef->RESCTLOCDR->CTLOCHOSPITALDR) pues es posible que el comportamiento sea diferente dependiendo esta variable (por cercanía al centro de atención por ejemplo).

Nota: Considerar si excluir del análsis citas canceladas (X) u otros estados como Transferido (T) pues son citas que se mostrarán como "NoShow" pero no se puede atribuir eso a un comportamiento del paciente necesariamente. Para este ejemplo se han considerado, pero a valor "0", es decir NoShow = False.

```
import pandas as pd
import jaydebeapi as jdbc
import time
import pyodbc as odbc
import irisnative
```

Función Diagnósticos

Tendremos funciones para determinar si el paciente está diagnosticado con Huipertensión, Diabetes, Alcoholismo o alguna restricción física (invalidez).

La función recibirá un arreglo con los diagnósticos del paciente al momento de la cita (filtrados por fecha) para retornar un 0 (False) o un 1 (True). Basta con que uno de los diagnósticos sea HTA para saber que el paciente en ese momento es hipertenso.

```
def is_hta (dxlist):
    ishta = 0
    htalist = ('I10', 'I11','I12', 'I13', 'I14', 'I15')
    for i in dxlist:
        if i in htalist:
            ishta = 1
    return ishta

def is_dbt (dxlist):
```

```
    isdbt = 0
    dbtlist = ('E10', 'E11','E12', 'E13', 'I14')
    for i in dxlist:
        if i in dbtlist:
            isdbt = 1
    return isdbt

def is_disabilited (dxlist):
    isdisabilited = 0
    disabilitedlist = ('Z73.6','1')
    for i in dxlist:
        if i in disabilitedlist:
            isdisabilited = 1
    return isdisabilited

def is_alcoholism (dxlist):
    isalcoholism = 0
    alcoholismlist = ('F10','1')
    for i in dxlist:
        if i in alcoholismlist:
            isalcoholism = 1
    return isalcoholism

def listdxbypatid (patid):
listadxsql = ()
if patid:
    idpaciente = patid
    sqlpatdx = """
    select %nolock
    MR_DIAGNOS.MRDIA_ICDCode_DR->MRCID_Code,
    MR_DIAGNOS.MRDIA_DATE
    from    SQLUser.MR_ADM, SQLUser.MR_DIAGNOS
    where   MR_ADM.MRADM_RowId IN (SELECT PAADM_MainMRADM_DR
    FROM  PA_Adm
    WHERE PAADM_PAPMI_DR = """ + str(idpaciente) + """ )
    and MR_ADM.MRADM_RowId = MR_DIAGNOS.MRDIA_MRADM_ParRef
    """
    listadxsql = pd.read_sql(sqlpatdx, conn)
return listadxsql
```

Preparar la Conexión

Preparamos la conexión utilizando la librería ***jaydebeapi*** y el conector que corresponde al ambiente a utilizar. En este caso, se trata de un ambiente sobre HealthShare, por lo que usaremos el JDBC de Cache.

```
# Establecemos nuestras variables de conexión (host, port, namespace)
envhost = "localhost"
envport = 1983
envns = "TRAK"

dbuser = "superuser"
dbaccess = "SYS"

url = "jdbc:Cache://" + envhost + ":" + str(envport) + "/" + envns
driver = 'com.intersys.jdbc.CacheDriver'
irisdriver = 'com.intersystems.jdbc.IRISDriver'
```

```
user = dbuser
password = dbaccess
# Para este ejemplo, el jar para Cache se encuentra bajo la carpeta "resources"
# ruta relativa a la ubicación de este notebook
#jarfile = "resources/cachejdbc.jar"
jarfile = "/Users/ArielArias/cachejdbc.jar"
conn = jdbc.connect(driver, url, [user, password], jarfile)
curs = conn.cursor()
```

Asignamos la consulta en una variable, para poder modificar posteriormente si es necesario. De esta forma además podemos utilizar asignación en múltiples líneas para tener una mejor vosión de lo que estamos extrayendo desde TrakCare.

```
sql_str = """Select TOP 3500
APPT_PAPMI_DR->PAPMI_PAPER_DR PatientID,
APPT_RowId ApptID,
APPT_DateComp ApptDate,
APPT_PAPMI_DR->PAPMI_Sex_DR->CTSEX_Gender Gender,
{fn DAYOFWEEK(APPT_DateComp)} ScheduledWeekDay,
{fn WEEK(APPT_DateComp)} ScheduledWeek,
DATEDIFF('dd',APPT_BookedDate, APPT_DateComp) WaitingDays,
DATEDIFF('YY',APPT_PAPMI_DR->PAPMI_PAPER_DR->PAPER_DOB,APPT_DateComp) Age,
CASE APPT_Confirmation
WHEN 'Y' THEN 1
ELSE 0
END SMSreceived,
CASE APPT_Status
WHEN 'P' THEN 1
WHEN 'A' THEN 0
WHEN 'X' THEN 0
WHEN 'I' THEN 0
WHEN 'N' THEN 1
WHEN 'T' THEN 0
WHEN 'H' THEN 0
WHEN 'C' THEN 0
WHEN 'S' THEN 0
WHEN 'D' THEN 0
ELSE APPT_Status
END Noshow
from RB_Appointment
Where
APPT_DateComp BETWEEN to_date('01/01/2021','dd/mm/yyyy') AND NOW()
AND APPT_AS_ParRef->AS_RES_ParRef->RES_CTLOC_DR->CTLOC_HOSPITAL_DR IN (12129)"""
```

Ejecutar la consulta (DataFrame)

Utilizando la librerías ***pandas*** ejecutamos la consulta SQL y la dejamos en un DataFrame para posterior manipulación. Nos falta aún las columnas relacionadas a diagnósticos del paciente a la fecha de la cita (por ahora no consideraremos TODOS los diagnósticos del paciente, sólo los confirmados y que fueron ingresados previo a la fecha del compromiso.

```
start = time.time()
data = pd.read_sql(sql_str, conn)
end = time.time()
elapsedtime = end - start
```

```
# Sólo para tener una referencia veremos el tiempo de ejecución de la consulta
# Esto ayudará a evaluar cuántos registros se puede procesar
# O planificar una ejecución por períodos de tiempo más reducidos
print("Tiempo de Ejecución SQL: ",int(elapsedtime), " Segundos")

# Agregamos las columnas de Dx;  con valores negativos por ahora
data["Hypertension"] = 0
data["Diabetes"] = 0
data["Alcoholism"] = 0
data["Handicap"] = 0

# Verificamos nuestro dataframe
data.head()
```

	PatientID	ApptID	ApptDate	Gender	ScheduledWeekDay	ScheduledWeek	WaitingDays	Age	SMSreceived	Noshow	Hypertension	Diabetes	Alcoholism	Handicap
0	598205	11683||11250||1	2021-01-21	F	5	4	0	27	0	0	0	0	0	0
1	598205	11683||11804||1	2021-01-28	F	5	5	1	27	0	1	0	0	0	0
2	598205	11683||13466||1	2021-02-18	F	5	8	0	27	0	0	0	0	0	0
3	598205	11683||15128||1	2021-03-11	F	5	11	0	27	0	0	0	0	0	0
4	598205	11683||15682||1	2021-03-18	F	5	12	0	27	0	0	0	0	0	0

Recorremos el Arreglo y traemos la lista de diagnósticos para cada paciente. Para optimizar el tiempo, verificamos si ya tenemos los diagnósticos del paciente teniendo una lista con los **PatientID**

```
start = time.time()
patient = []
## is_hta
## is_dbt
## is_disabilited
## is_alcoholism
for index, row in data.iterrows():
    if row['PatientID'] not in patient:
        patient.append(row['PatientID'])
        patappt = (row['ApptID'], row['PatientID'], row['ApptDate'])
        listpatdx = listdxbypatid(row['PatientID'])
    # Ya tenemos los diagnósticos del Paciente;
    # ahora filtramos sólo los previos a la cita y los dejamos en una lista:
    if len(listpatdx):
        low_apptdate = listpatdx[listpatdx["MRDIA_Date"] <= patappt[2]]
        if len(low_apptdate):
            # Ahora necesitamos saber si el paciente "ES" o "ERA" Hipertenso para la
fecha de la cita:
```

```
            diaghta = is_hta(low_apptdate.MRCID_Code)
            diagdbt = is_dbt(low_apptdate.MRCID_Code)
            diagdis = is_disabilited(low_apptdate.MRCID_Code)
            diagalcohol = is_alcoholism(low_apptdate.MRCID_Code)
        else:
            diaghta = 0
            diagdbt = 0
            diagdis = 0
    else:
        diaghta = 0
        diagdbt = 0
        diagdis = 0
    # Modificamos la información en la fila correspondiente
    # Sabemos que por ApptID habrá sólo un registro:
    data.loc[data.ApptID == row['ApptID'],'Hypertension'] = diaghta
    data.loc[data.ApptID == row['ApptID'],'Diabetes'] = diagdbt
    data.loc[data.ApptID == row['ApptID'],'Handicap'] = diagdis
    data.loc[data.ApptID == row['ApptID'],'Alcoholism'] = diagalcohol

end = time.time()

print(end - start)
```

Cerramos la conexión, de momento no la usaremos.

```
conn.close()
```

LLEVAR LOS REGISTROS A UNA INSTANCIA CON ML INTEGRADO

En este ejemplo, no podemos usar el mismo servidor pues en su versión no incluye la función de ML; así es que usaremos una instancia con IRIS for Health que incorpora ML.

Dado que la librería de python ***jaydebeapi*** soporta sólo un dirver conexión por ejecución, no podemos usar el mismo método pues habrá un error al intentar usarla para conectar IRIS (posterior a conectar Caché).

Para seguir con el ejemplo y no perder el DataFrame obtenido, vamos a crear una conexión a IRIS utilizando ***ODBC***

Usaremos la misma instancia del ejemplo que hemos usado hasta ahora, es decir el NameSpace *FHIRSERVER*

```
dsn = 'IRIS FHIRML'
server = 'localhost' #IRIS server container or the docker machine's IP
port = '32782' # or 8091 if docker machine IP is used
database = 'FHIRSERVER'
username = 'SUPERUSER'
password = 'SYS'

irisconn = odbc.connect('DSN='+dsn+';')
irisconn.setencoding(encoding='utf-8')
iriscurs = irisconn.cursor()

modelName = "TrakNoShow"

fhirsql = """Select
Age, Alcoholism, Diabetes,
```

```
Gender, Handicap, Hypertension,
Noshow, SMSreceived, ScheduledWeek,
ScheduledWeekDay, WaitingDays
FROM PackageSample.""" + modelName + "MLRow"

# Copiamos la tabla NoShowMLRow a nuestra TrakNoShowMLRow:
sqlcreate = "CREATE TABLE PackageSample." + modelName + "MLRow ("
sqlcreate = sqlcreate + "Gender CHAR(30),"
sqlcreate = sqlcreate + "ScheduledWeekDay INT,"
sqlcreate = sqlcreate + "ScheduledWeek INT,"
sqlcreate = sqlcreate + "WaitingDays INT,"
sqlcreate = sqlcreate + "Age INT,"
sqlcreate = sqlcreate + "Hypertension BIT,"
sqlcreate = sqlcreate + "Diabetes BIT,"
sqlcreate = sqlcreate + "Alcoholism BIT,"
sqlcreate = sqlcreate + "Handicap BIT,"
sqlcreate = sqlcreate + "SMSreceived BIT,"
sqlcreate = sqlcreate + "Noshow BIT)"
```

Ejecutamos la creción de la tabla:

```
iriscurs.execute(sqlcreate)
iriscurs.commit()
```

Inserción del DataFrame TrakCare

Para hacer una inserción directa de los datos, haremos una copia del DataFrame para eliminar luego algunas columnas que no usaremos de momento; esto sólo para hacer un INSERT más directo al momento de recorrer el **DataFrame**.

```
sqltest = "INSERT INTO PackageSample." + modelName + "MLRow"
sqltest = sqltest + " (Gender,ScheduledWeekDay,ScheduledWeek,WaitingDays,"
sqltest = sqltest + "Age,SMSreceived,Noshow,Hypertension,Diabetes,Alcoholism,Handicap
) "
sqltest = sqltest + "values(?,?,?,?,?,?,?,?,?,?,?)"

for index, row in data.iterrows():
    params = [[row.Gender,row.ScheduledWeekDay,row.ScheduledWeek,row.WaitingDays,row.
Age,row.SMSreceived,row.Noshow,row.Hypertension,row.Diabetes,row.Alcoholism,row.Handi
cap]]
    iriscurs.executemany(sqltest,params)

iriscurs.commit()
```

Preparación del Modelo

Para efecto de visualización, vamos a trabajar con variables para nombrar nuestro modelo, y definir con cuántos datos vamos a entrenar y cuántos vamos a utilizar para validar.

```
datasize = len(irisdata)
modelName = "TrakNoShow"
predictingCol = "Noshow"
# si requiere reiniciar el modelo, se elimina las tablas creadas:
```

```
sqldropcross = "DROP TABLE PackageSample." + modelName + "MLRowCrossValidation"
sqldroptest = "DROP TABLE PackageSample." + modelName + "MLRowTest"
sqldroptrain = "DROP TABLE PackageSample." + modelName + "MLRowTraining"
sqldropval = "DROP TABLE PackageSample." + modelName + "MLRowValidation"
sqldropmodel = "DROP MODEL " + modelName + "Model"

# definimos con qué porcentaje de los datos realizaremos el crossvalidation:
cvLen = int(datasize * 0.9)
# Y el resto para testing:
testLen = datasize - cvLen
# tamaño de los datos para entrenamiento:
trainLen = int(cvLen * 0.75)
validationLen = cvLen - trainLen
# Crearemos modelo con CrossValidation:
sqlcrossval = "CREATE TABLE PackageSample." + modelName + "MLRowCrossValidation AS "
sqlcrossval = sqlcrossval + "SELECT * FROM PackageSample." + modelName + "MLRow WHERE
 Id IN ("
sqlcrossval = sqlcrossval + "SELECT TOP " + str(cvLen) + " idx "
sqlcrossval = sqlcrossval + "FROM community.randrange(1, " + str(datasize+1)  + ") "
sqlcrossval = sqlcrossval + "ORDER BY randValue)"

# Creamos la tabla para pruebas del modelo
sqltest = "CREATE TABLE PackageSample." + modelName + "MLRowTest AS "
sqltest = sqltest + "SELECT TOP " + str(testLen) + " * FROM PackageSample." + modelNa
me + "MLRow WHERE Id NOT IN ("
sqltest = sqltest + "SELECT Id FROM PackageSample." + modelName + "MLRowCrossValidati
on)"

# Creamos tabla para entrenamiento:
sqltrain = "CREATE TABLE PackageSample." + modelName + "MLRowTraining AS "
sqltrain = sqltrain + "SELECT * FROM PackageSample." + modelName + "MLRowCrossValidat
ion WHERE Id IN ("
sqltrain = sqltrain + "SELECT TOP " + str(trainLen) + " idx "
sqltrain = sqltrain + "FROM community.randrange(1, " + str(datasize+1) + ") "
sqltrain = sqltrain + "ORDER BY randValue)"

# Tabla para validación:
sqlval = "CREATE TABLE PackageSample." + modelName + "MLRowValidation AS "
sqlval = sqlval + "SELECT TOP " + str(validationLen) + " * "
sqlval = sqlval + "FROM PackageSample." + modelName + "MLRowCrossValidation "
sqlval = sqlval + "WHERE Id NOT IN ("
sqlval = sqlval + "SELECT Id FROM PackageSample." + modelName + "MLRowTraining)"

# Para creación del modelo:
sqlcreatemodel = "CREATE MODEL "+ modelName + "Model PREDICTING (" + predictingCol +
") "
sqlcreatemodel = sqlcreatemodel + "FROM PackageSample." + modelName + "MLRowTraining"

# Para entrenar modelo:
sqltrainmodel = 'TRAIN MODEL ' + modelName + 'Model USING {"seed": 1}'

# Para validar modelo:
sqlvalmodel = "VALIDATE MODEL " + modelName + "Model FROM PackageSample." + modelName
 + "MLRowValidation"
```

Podemos ejecutar una a una las sentencias desde nuestro notebook para crear el modelo, y luego entrenarlo.

```
# Creamos tabla CrossValidation:
iriscurs.execute(sqlcrossval)
iriscurs.commit()

# Creamos tabla Test:
iriscurs.execute(sqltest)
iriscurs.commit()

# Creamos tabla para entrenamiento:
iriscurs.execute(sqltrain)
iriscurs.commit()

# Creamos tabla para validacion:
iriscurs.execute(sqlval)
iriscurs.commit()

# Creamos el modelo:
iriscurs.execute(sqlcreatemodel)
iriscurs.commit()

# Entrenamos el modelo:
iriscurs.execute(sqltrainmodel)
iriscurs.commit()

# Validamos el modelo:
iriscurs.execute(sqlvalmodel)
iriscurs.commit()
```

Entrenado el modelo, podemos utilizarlo para conocer la predicción, ya sea que enviamos datos desde otro origen o podemos utilizar consultas directamente sobre el modelo. Un ejemplo:

```
# TrakNoShowModel
# Probar consultas hacia el modelo:

sqlpredict = "SELECT top 10 PREDICT(TrakNoShowModel) AS PredictedNoshow, Noshow AS Ac
tualNoshow FROM PackageSample.TrakNoShowMLRowTest"

predictdata = pd.read_sql(sqlpredict, irisconn)
predictdata
```

	PredictedNoshow	ActualNoshow
0	False	False
1	True	True
2	False	False
3	False	False
4	False	False
5	False	False
6	False	False
7	False	False
8	False	False
9	True	False

ALGUNOS PROBLEMAS ENCONTRADOS

Durante la preparación de este documento se debió ajustar algunos parámetros y consultas para poder terminarlo. Se detalla a continuación para posible solución al replicar los pasos en otro ambiente:

- Se ajustó la conexión a IRIS para forzar utilización de encoding UTF8; al no hacerlo, se detectó que el campo "Gender" quedaba vacío.
- Problema de Performance al consultar los diagnósticos del paciente de forma iterativa; por eso se ajustó la consulta para traer todos los diagnósticos del paciente con fecha de ingreso, y luego con Python filtrar según fecha de la cita.
- ***jaydebeapi*** soporta una conexión por ejecución; por esto se trabajó la conexión a TrakCare (sobre Caché) con esta librería y se utilizó ***pyodbc*** para conectar IRIS. Se puede hacer lo contrario, ODBC para Caché y JDBC para IRIS.

TO DO

Algunas mejoras a la aplicación de este ejemplo, y que pueden estar disponible en nuevos post:

1. Se puede trabajar estos mismos pasos, direcamente desde IRIS, aprovechando aún más sus propiedades y facilidad para acceder a los datos. Se hizo como notebook sólo para ejemplificar que se puede traer varios origenes de datos a IRIS para crear un modelo de Machine Learning, entrenarlo, y validarlo.

2. Tener estos pasos en una clase de IRIS, permitirá además operativizar el modelo, por medio de webservice como el ejemplo disponible en ***readmision demo***

3. Contar con una carga programada desde el origen que permita incluso actualizar datos, por ejemplo, del estado de cita. Esto permitirá por ejemplo, construir un cubo y posterior cuadro de mando que permita identificar a los pacientes con alta posibilidad de no asistir para reforzar el contacto.

#Bases de datos #Machine learning #ODBC #InterSystems IRIS for Health #TrakCare

URL de fuente:https://es.community.intersystems.com/post/utilizar-modelo-de-irisfhirml-noshow-teniendo-origen-de-datos-trakcare-2017