Trabajo Final – Modulo 5

July 10, 2022

Integrantes: Manuel Hernandez Martinez , Luis Moreno Diaz, Wilmer Urango Narvaez, Ivan Alvarez Gomez y Luis Durango Suarez.

Breve descripción: La base de datos a explorar se llama **"mdl_informes_2022_1_user_loggedin"**, contiene información del loggedin de los estudiantes de la Universidad de Córdoba en la plataforma Cintia, entre las variables mas relevantes con las que se cuenta en esta data estan **usr_username:Usuario** , **timecreated_unix: Fecha de conexión** y **prog_programa: Programa Academico**, tambien se encuenta con otras variables, pero no resultan muy relevantes, bien sea por tener valores unicos, la auxencia de datos o porque no presentan una caracteristica relevante en temas de interpretación.

1 Cargue su dataset, limpio, e imprima los 10 primeros registros.

En la data se encontro 21 variables que tenian todos los valores faltantes o tenian el mismo valor en toda la columna, para estas variables se verificara dicho supuesto y posteriormte se eliminaron de la base de datos, tambien se escluyeron 7 variables que no se utilizaron en la descripción de nuestros datos, ademas dado que l **timecreated_unix** no tenia el formato fecha, se procede a crear una nueva variable la cual se llamo **timecreated** y tiene el formato fecha y hora.

[1]:
```
# Importando modulos de análisis
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
from datetime import datetime
import numpy as np
import json
import plotly
from datetime import date
import holidays
import warnings
```

[2]:
```
# Leyendo los datos
Loggedin = pd.read_csv('data/mdl_informes_2022_1_user_loggedin.csv',sep=';')
```

[3]:
```
```

```
Loggedin["timecreated"]=pd.to_datetime(Loggedin["timecreated_unix"],unit="s")
Variables=np.
 ↪array(["eventname","component","action","target","objecttable","crud",␣
 ↪"edulevel",
                    "courseid"␣
 ↪,"relateduserid","origin","realuserid","cur_id","cur_idnumber",
                   ␣
 ↪"cur_shortname","cur_fullname","cur_category","cur_startdate","cur_enddate",
                   ␣
 ↪"prog_id","prog_idnumber","fac_idnumber","fac_facultad","id","userid","other",
                    "timecreated_unix","ip","usr_id","usr_idnumber"])
Loggedin.drop(columns=Variables, inplace=True)
# reindexar el dataframe
Loggedin.index = Loggedin.timecreated
```

[4]:
```
Loggedin.head(10)
```

[4]:

timecreated	usr_username	prog_programa	\
2022-03-22 05:03:34	1067880829	Departamento De Geografía Y Medio Ambien	
2022-03-22 05:13:10	1193517912	BIOLOGÍA	
2022-03-22 05:26:07	1005675329	Ingeniería Agronómica	
2022-03-22 05:30:06	1003193750	QUÍMICA	
2022-03-22 05:31:12	1062675102	LIC EN CIENCIAS NATURALES Y EDU AMBIENTA	
2022-03-22 05:44:06	1003432509	QUÍMICA	
2022-03-22 05:48:10	1003070670	Ingeniería Mecánica	
2022-03-22 06:10:13	1003362139	INGENIERÍA INDUSTRIAL	
2022-03-22 06:10:41	1067846685	Ingeniería Ambiental	
2022-03-22 06:21:42	1003072173	Adminis. en Finanzas y Negocios Internac	

timecreated	timecreated
2022-03-22 05:03:34	2022-03-22 05:03:34
2022-03-22 05:13:10	2022-03-22 05:13:10
2022-03-22 05:26:07	2022-03-22 05:26:07
2022-03-22 05:30:06	2022-03-22 05:30:06
2022-03-22 05:31:12	2022-03-22 05:31:12
2022-03-22 05:44:06	2022-03-22 05:44:06
2022-03-22 05:48:10	2022-03-22 05:48:10
2022-03-22 06:10:13	2022-03-22 06:10:13
2022-03-22 06:10:41	2022-03-22 06:10:41
2022-03-22 06:21:42	2022-03-22 06:21:42

La base final tiene valores diferentes en todas sus columnas, pero solo se considerara **usr_username** y **timecreated**, para el ejercicio de este trabajo, sin embargo en el entregable final si se incluirá la variable **prog_program**, dado que es la variable más relevante en la descripción de la data.

2 Realice un pre-procesamiento de datos principalmente enfocado en estos puntos:

2.1 Ingeniería de características

Para nuestra data fue necesario realizar modificaciones en la la variable **prog_programa**, se muestra un breve ejemplo de cómo se realizó la codificación, sin embargo se omite hacer el cálculo completo dado que esta variable no se tendrá en cuenta en el modelo a realizar, pero fue necesario realzar las medicaciones para el dashboard final

[5]:
```
Loggedin=Loggedin.replace({'prog_programa': {'Departamento De Geografía Y Medio␣
↪Ambien':'Dpto de GeografIa y Medio Ambiente',
            'BIOLOGÍA':'Biología','QUÍMICA':'Química',
            'LIC EN CIENCIAS NATURALES Y EDU AMBIENTA':'Lic en Ciencias␣
↪Naturales y Edu Ambienta',
            'INGENIERÍA INDUSTRIAL':'Ingeniería Industrial',
             'Adminis. en Finanzas y Negocios Internac':'Administración en␣
↪Finanzas y Negocios Internacionales'}})
```

2.2 Codificación LabelEncoding o OneHotEncoding

Se crea la base de datos del total de Loggedin por horas, adicionalemte se elimina la información del primer y ultimo dia de la data por no tener la información del total de horas.

[6]:
```
loggedin_day = Loggedin.usr_username.resample('H').count()[19:-5]
```

[7]:
```
loggedin_day
```

[7]:
```
timecreated
2022-03-23 00:00:00     65
2022-03-23 01:00:00     59
2022-03-23 02:00:00     70
2022-03-23 03:00:00    114
2022-03-23 04:00:00     60
                      ...
2022-06-10 19:00:00    403
2022-06-10 20:00:00    460
2022-06-10 21:00:00    513
2022-06-10 22:00:00    483
2022-06-10 23:00:00    326
Freq: H, Name: usr_username, Length: 1920, dtype: int64
```

Ahora se codifican y se transforman los datos a utilizar en el modelo 2.

[8]:
```
warnings.filterwarnings('ignore')
df = pd.concat([loggedin_day], axis=1)
df['dayofweek'] = df.index.dayofweek #Dia de la semana
```

```
df['month'] = (df.index.day_of_year-80)//30+1 #Mes (Numero de meses contados en␣
 ↪30 dias)
df['week'] = df.index.weekofyear-11 #Numero de la semana del semestre
df['hour'] = df.index.hour # Hora del dia
df['court'] = (df.index.day_of_year-80)//46+1 # corte academico en el semestre
df['dayofsemester'] = df.index.day_of_year-80 # Dia en el semestre
df['dayofmonth'] = df['dayofsemester'] -30*(df['month']-1) # Dia en el mes
df['dayofcuort'] = df['dayofsemester'] -46*(df['court']-1) # Dia en el corte
df['serie'] = df.index.date
co_holidays = holidays.CO() # Dias festuvos
df['holidays'] = np.where(df['serie'].isin(co_holidays),'1','0') # 1 si es␣
 ↪festivo
df = df.rename(
    columns={'usr_username':'loggedin'}
)
df.head()
```

[8]:

timecreated	loggedin	dayofweek	month	week	hour	court	dayofsemester	dayofmonth	dayofcuort	serie	holidays
2022-03-23 00:00:00	65	2	1	1	0	1	2	2	2	2022-03-23	0
2022-03-23 01:00:00	59	2	1	1	1	1	2	2	2	2022-03-23	0
2022-03-23 02:00:00	70	2	1	1	2	1	2	2	2	2022-03-23	0
2022-03-23 03:00:00	114	2	1	1	3	1	2	2	2	2022-03-23	0
2022-03-23 04:00:00	60	2	1	1	4	1	2	2	2	2022-03-23	0

2.3 Partición del conjunto de datos en train-test Split

Ahora procedemos a dividir nuestra data en nuestro caso se considerará test los últimos 7 días, esto con el fin de estimar la última semana, de la muestra.

[9]:
```
# Train test split
from sklearn.model_selection import train_test_split
train, test = train_test_split(loggedin_day, test_size = 0.087, random_state =␣
↪42,shuffle=False)
```

[10]:
```
X_train,X_test,y_train,y_test =␣
↪train_test_split(df[['dayofsemester','court','dayofcuort','month',
                                                    ␣
↪'dayofmonth','week','dayofweek',
                                                       'hour','holidays']],
                                                  df['loggedin'],
                                                  test_size=0.087,random_state =␣
↪90,shuffle=False)
```

Se divide la base en train y test dejando en ambos casos los últimos 7 días, con el objetivo de realizar predicciones en la última semana, adicionalmente como se quiere realizar un modelo RandomForestRegressor, se establece la estructura pero conservando la información de una serie de tiempos.

3 Entrene al menos 2 modelos de acuerdo alproblema que usted tenga.

3.1 Modelo 1. Serie de Tiempo Modelo Sarima

El primer modelo a considerar es una serie de tiempo con modelo SARIMA.

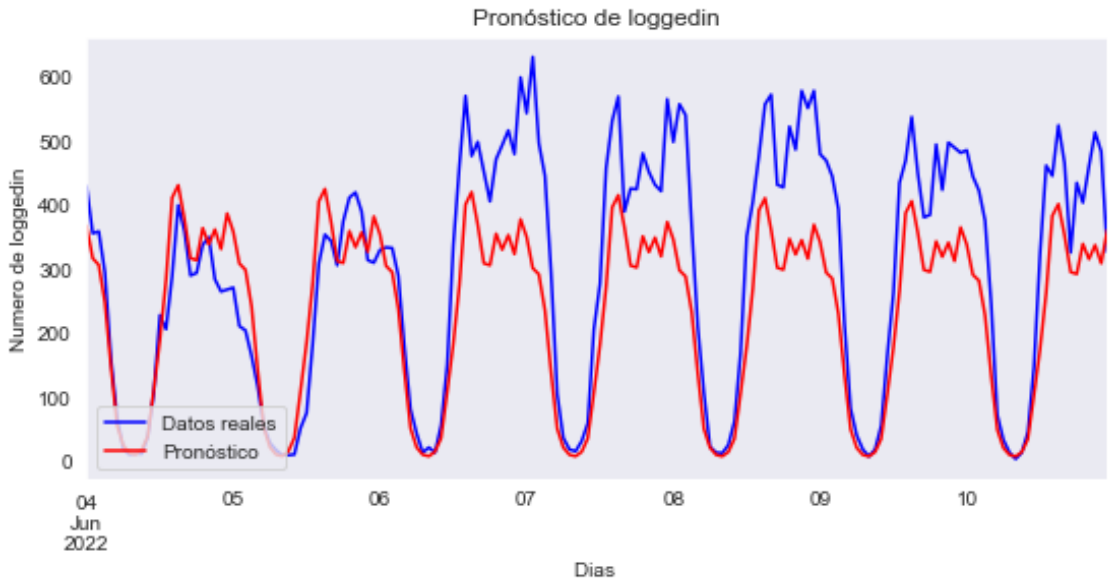
[11]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# definir conjunto de datos
x = train
# instanciar modelo
sarima_model = SARIMAX(x, order=(1,0,1), seasonal_order=(1, 0, 1, 24))
# ajustar modelo
results = sarima_model.fit()
# mirar el AIC
results.aic
```

[11]:
```
18739.418010830534
```

[12]:
```
pred_test = results.get_forecast(steps=24*7).predicted_mean
fig, ax = plt.subplots(figsize=(9, 4))
test.plot(color='blue')
pred_test.plot(color='red')
plt.grid()
ax.legend();
plt.title('Pronóstico de loggedin')
plt.ylabel('Numero de loggedin')
```

```
plt.xlabel('Dias ')
plt.legend(('Datos reales', 'Pronóstico'),
           loc='lower left');
```



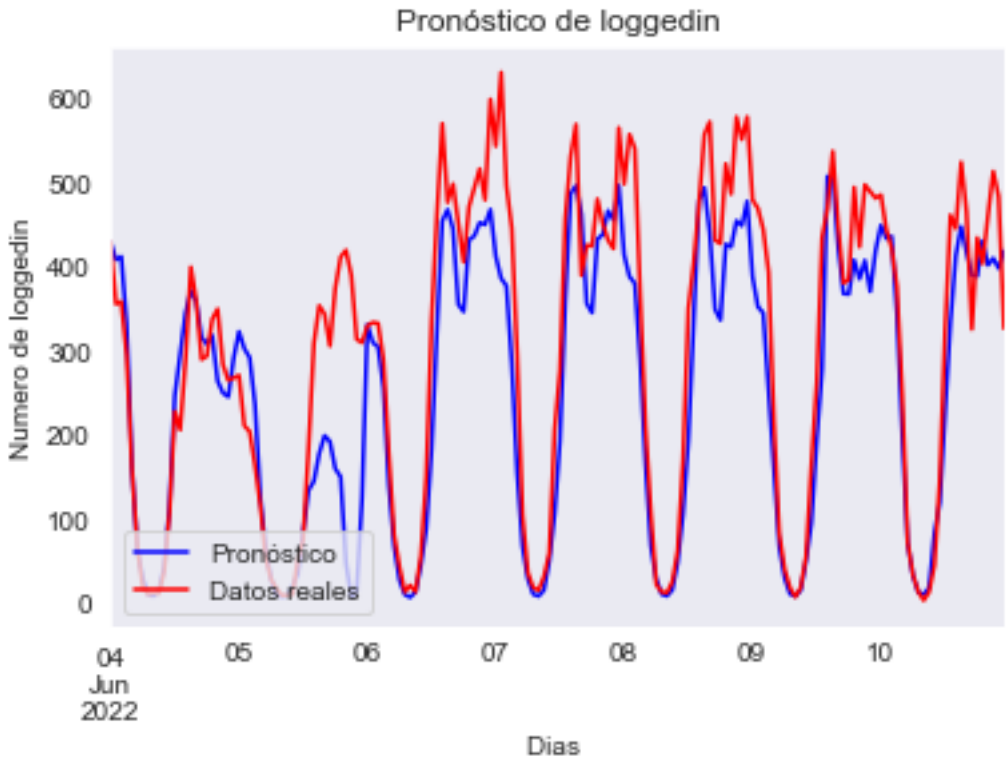
3.2 Modelo 2: Modelo de Regresión Random Forest Regressor

Para el segundo modelo se creó la estructura con los datos de la serie de las variables que tendría el semestre académico

[13]:
```
from sklearn.ensemble import RandomForestRegressor
rg=RandomForestRegressor()
rg.fit(X_train,y_train)
```

[13]:
```
RandomForestRegressor()
```

[14]:
```
y_pred_test=pd.DataFrame(rg.predict(X_test))
y_pred_test.index = y_test.index
y_pred_test.plot(color='blue')
y_test.plot(color='red')
plt.grid()
plt.title('Pronóstico de loggedin')
plt.ylabel('Numero de loggedin')
plt.xlabel('Dias ')
plt.legend(('Pronóstico','Datos reales'),
           loc='lower left');
```



Los dos modelos están mostrando predicciones cercanos a sus valores reales, por lo que a primera impresión se considera son resultados acertados.

4 Presente métricas de desempeño de los modelos que entreno y realice una conclusión de lo que observa en caso.

Se calcula para cada modelo el R2 y el MSE.

4.1 Modelo 1. Serie de Tiempo Modelo Sarima

[15]:
```
# Metricas de error
from sklearn.metrics import mean_squared_error, r2_score
# MSE
print("MSE: %.2f" % mean_squared_error(test, pred_test))
# R2
print("Metrica de R2:" ,r2_score(test, pred_test))
```

```
MSE: 12379.89
Metrica de R2: 0.6499799006791949
```

4.2 Modelo 2. Modelo de Regresión Random Forest Regressor

[16]:
```
y_pred_test=pd.DataFrame(rg.predict(X_test))
# Metricas de error
# MSE
print("MSE: %.2f" % mean_squared_error(y_test, y_pred_test))
# R2
print("Metrica de R2:" ,r2_score(y_test, y_pred_test))
```

```
MSE: 8224.24
Metrica de R2: 0.7674738224781101
```

El Modelo de regresión random forest regressor tiene mejor MSE y mejor R2 que la serie de tiempo con modelo SARIMA, ambos modelos presentan estimaciones cercanas a los datos reales, sin embargo el primer modelo tiende a subestimar las predicciones, mientras que el segundo se puede ver que las esta sobrestimado, en este caso por las métricas se recomienda quedarse con el segundo modelo.

5 Realice una búsqueda de hiper-parametros ("la malla") de acuerdo con el modelo que esté empleando.

Para el caso de la serie de tiempo se consideraron varios modelos con estructuras diferentes en los parametros finalmete el modelo que se quedo fue el del menor AIC, para el cual se ilustra nuevamente las estiamciones resultados de aplicarr el modelo.

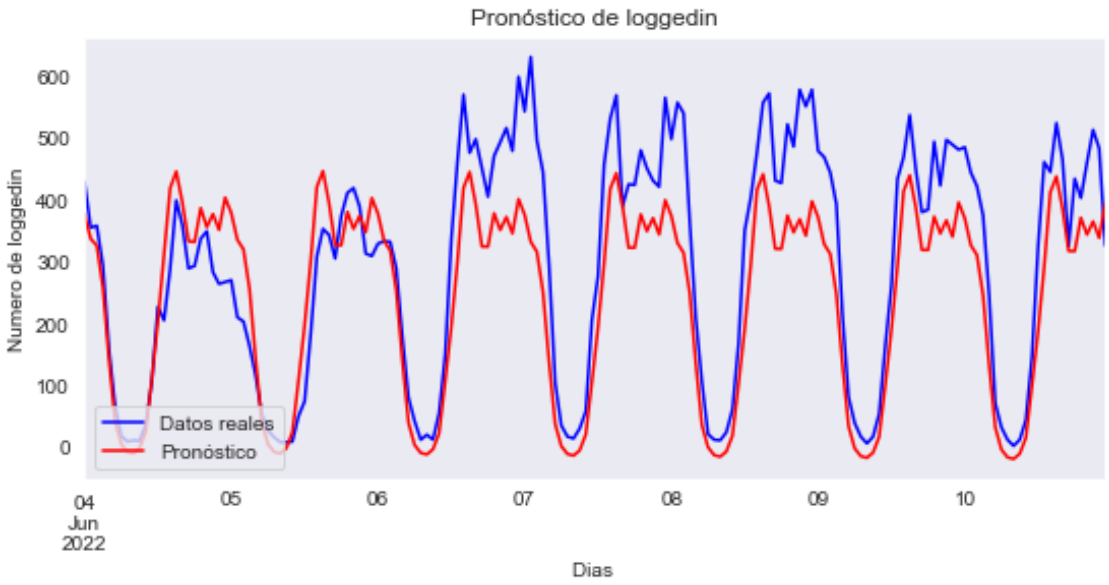
[17]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# definir conjunto de datos
x = train
# instanciar modelo
sarima_model = SARIMAX(x, order=(1,1,1), seasonal_order=(1, 1, 1, 24))
# ajustar modelo
results = sarima_model.fit()
# mirar el AIC
results.aic
```

[17]:
```
18459.123883353168
```

[18]:
```
pred_test = results.get_forecast(steps=24*7).predicted_mean
fig, ax = plt.subplots(figsize=(9, 4))
test.plot(color='blue')
pred_test.plot(color='red')
plt.grid()
ax.legend();
plt.title('Pronóstico de loggedin')
plt.ylabel('Numero de loggedin')
plt.xlabel('Dias ')
plt.legend(('Datos reales', 'Pronóstico'),
```

```
            loc='lower left');
```



Para el segundo modelo se incluyo la información de festivos y ademas se mejoraron las estimaciones al incluir información de periodos y dias de los periodos academicos, pero estos cambios fueron realizads directamente en el punto dos. En general se observo estiamciones mas cercanas al agregar los paramertros optimos.

6 Presente las métricas de desempeño de los modelos con sus parámetros óptimos.

Se calcula nuevamente el MSE Y R2

[19]:
```
# Metricas de error
from sklearn.metrics import mean_squared_error, r2_score
# MSE
print("MSE: %.2f" % mean_squared_error(test, pred_test))
# R2
print("Metrica de R2:" ,r2_score(test, pred_test))
```

```
MSE: 9787.60
Metrica de R2: 0.7232725196522533
```

Si mejoraron las estimaciones al incluir los parámetros óptimos, al incluir una estructura en la serie de tiempo con parámetros optimas las estimaciones mejoran dado que se acercan más a la realidad de la serie de tiempos

7 Elija el mejor modelo de los que optimizó. Ahora, con esa configuración, entrene el modelo con todo el conjunto de datos

[20]:
```
#Todos los datos
DatosFinal=df[['dayofsemester','court','dayofcuort','month',
     'dayofmonth','week','dayofweek',
     'hour','holidays','loggedin']]
from sklearn.ensemble import RandomForestRegressor
rg=RandomForestRegressor()
rg.fit(DatosFinal[['dayofsemester','court','dayofcuort','month',
     'dayofmonth','week','dayofweek',
     'hour','holidays','loggedin']],DatosFinal['loggedin'])
```

[20]:
```
RandomForestRegressor()
```

8 Guarde el modelo en formato pkl. Este debe adjuntarlo.

[32]:
```
## UTILIZANDO pickle

import pickle
from sklearn import svm
################################
## GUARDAMMOS MODELO con pickle
################################
# Escribir modelo
with open('model.pkl','wb') as f:
    pickle.dump(rg,f)
```

9 Guarde el dataset con todas las transformaciones de preprocesamiento que acabó de utilizar.

[30]:
```
# Creación DataFrame:
df_data = pd.DataFrame(DatosFinal)

# Guarda datos en CSV:
df_data.to_csv('DatosFinal.csv')
```

[ ]:
```

```