Tecnológico de Monterrey

TC2029 Ciencia y Analítica de Datos
DRA. María de la Paz Rico Fernández

Juan Pablo Bladinieres Marín del Campo A01793474
Gerardo Quiroga Nájera A00967999

Limpieza, Análisis, Visualización y K-Means en datos de Calidad del Agua en sitios de monitoreeo de aguas superficiales

Noviembre 2022

≫Introducción

45 Columnas

- Numéricas
- Calidades
- Binarias

```
numericas=['DBO_mg/L','DQO_mg/L','SST_mg/L','COLI_FEC_NMP_100mL','E_COLI_NMP_100mL','ENTEROC_NMP_100mL','OD_PORC','OD_PORC_SUP','OD_PORC_
for i in numericas:
    df[i]=df[i].replace({np.NAN:0})
    df[i]=df[i].replace({'<2':1.9,'<10':9.9,'<3':2.9,'<1':0.9})
    df[i]=df[i].astype("float")

calidades1=['CALIDAD_DBO','CALIDAD_DQO','CALIDAD_SST','CALIDAD_COLI_FEC','CALIDAD_E_COLI','CALIDAD_ENTEROC','CALIDAD_OD_PORC','CALIDAD_OD_
for i in calidades1:
    df[i]=df[i].replace({np.NAN:'Sin Medida','Excelente':'No Toxico ','Fuertemente contaminada':'Toxicidad alta','Contaminada':'Toxicidad
    df[i]=df[i].astype("string")

calidades2=['CALIDAD_TOX_V_15','CALIDAD_TOX_D_48','CALIDAD TOX_D_48_SUP','CALIDAD_TOX_D_48_FON','CALIDAD_TOX_FIS_SUP_15','CALIDAD_TOX_FIS_
for i in calidades2:
    df[i]=df[i].replace({np.NAN:'Sin Medida'})
    df[i]=df[i].astype("string")

binarias=['CUMPLE_CON_DBO','CUMPLE_CON_DQO','CUMPLE_CON_SST','CUMPLE_CON_CF','CUMPLE_CON_E_COLI','CUMPLE_CON_ENTEROC','CUMPLE_CON_OD','CU
for i in binarias:
    df[i]=df[i].replace({'ND':0,'NO':0,'SI':1})
    df[i]=df[i].astype("float")


df.describe(include='all')
```

3,479 registros

	SUBTIPO	LONGITUD	LATITUD	DBO_mg/L	CALIDAD_DBO	DQO_mg/L	CALIDAD_DQO	SST_mg/L	CALIDAD_SST	COLI_FEC_NMP_100mL	...	SEM
count	3479	3493.000000	3493.000000	4141.000000	4141	4141.000000	4141	4141.000000	4141	4.141000e+03	...	3493.
unique	23	NaN	NaN	NaN	5	NaN	5	NaN	5	NaN	...	
top	RIO	NaN	NaN	NaN	Sin Medida	NaN	Sin Medida	NaN	No Toxico	NaN	...	
freq	1478	NaN	NaN	NaN	1560	NaN	1560	NaN	1780	NaN	...	
mean	NaN	-100.359969	21.046992	10.495438	NaN	40.083925	NaN	86.043679	NaN	5.966397e+04	...	1.
std	NaN	6.122773	3.893696	52.075176	NaN	122.321070	NaN	407.284109	NaN	9.240891e+05	...	0.
min	NaN	-117.124030	14.534910	0.000000	NaN	0.000000	NaN	0.000000	NaN	0.000000e+00	...	1.
25%	NaN	-103.882310	18.396070	0.000000	NaN	0.000000	NaN	9.900000	NaN	0.000000e+00	...	1.
50%	NaN	-99.795530	20.148980	1.900000	NaN	9.900000	NaN	18.000000	NaN	2.100000e+02	...	2.
75%	NaN	-96.860230	22.828930	5.000000	NaN	35.750000	NaN	48.000000	NaN	4.600000e+03	...	3.
max	NaN	-86.732150	32.706500	1500.000000	NaN	2871.250000	NaN	9430.000000	NaN	2.419600e+07	...	3.

⋙Limpieza

```
df.dropna(how='any',inplace=True)

X = df.copy()
X.drop(binarias, axis=1,inplace=True)
#X.drop(calidades1, axis=1,inplace=True)
#X.drop(calidades2, axis=1,inplace=True)
X.drop(numericas, axis=1,inplace=True)
X.drop(localizacion, axis=1,inplace=True)
X.drop(['SUBTIPO','GRUPO'], axis=1,inplace=True)
X.drop('SEMAFORO',axis=1,inplace=True)
#X.drop(['Coordinates'], axis=1,inplace=True)
y = df['SEMAFORO'].copy()

print(X.shape)
print(y.shape)
```

```
(3479, 16)
(3479,)
```

Se debe tomar la decisión de como manejar los datos con categorias o con datos númericos. Por esta razón se decide usar los categoricos y se divide la base en X y Y para su modelaje los Y son los datos de salida en este caso nuestra variable de SEMÁFORO.

```
# Transformación completa de los resultados del analisis
path1='./Datos_de_calidad_del_agua_2020/Datos_de_calidad_del_agua_de_sitios_de_monitoreo_de_aguas_superficiales_2020.csv'
path2='./Datos_de_calidad_del_agua_2020/Datos_de_calidad_del_agua_de_sitios_de_monitoreo_de_aguas_subterraneas_2020.csv'
df=pd.read_csv(path1)

df.rename(mapper=df['CLAVE'],axis=0,inplace=True)

#df["Coordinates"] = list(zip(df.LONGITUD, df.LATITUD))
#df["Coordinates"] = df["Coordinates"].apply(Point)

df.drop(['CLAVE','TIPO','CONTAMINANTES','PERIODO','SITIO','ESTADO','MUNICIPIO','CUENCA','CUERPO DE AGUA','ORGANISMO_DE_CUENCA'],axis=1,inp
```

⋙Pipeline

```
#Transformer
pipeline = Pipeline(steps = [('impModa', SimpleImputer(strategy='most_frequent')),
                             ('OneHotE', OneHotEncoder())
                            ])

columnasTransformer = ColumnTransformer(transformers = [('transform', pipeline, X.columns)],
                                        remainder='passthrough')
```

```
# Division Train-Test
X.dropna(inplace=True)
X_tran = columnasTransformer.fit_transform(X)
#y_tran = columnasTransformer.fit_transform(np.ravel(y))

X_tv,X_test,y_tv,y_test = train_test_split(X_tran,y, test_size=.20, shuffle=True, random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_tv, y_tv, test_size=0.2, shuffle=True, random_state=42)

print(X_train.shape)
print(y_train.shape)
print(X_val.shape)
print(y_val.shape)
print(X_test.shape)
print(y_test.shape)
```

Análisis

```
# Random Tree Class
rtc=RandomForestClassifier(n_jobs=-1)

clfrtc = rtc.fit(X_train,y_train)

y_hat_val=clfrtc.predict(X_val)
y_hat_test=clfrtc.predict(X_test)

print(' VALIDACION RECALL: ',classification_report(y_val,y_hat_val))
print(' PRUEBA RECALL: ',classification_report(y_test,y_hat_test))
```

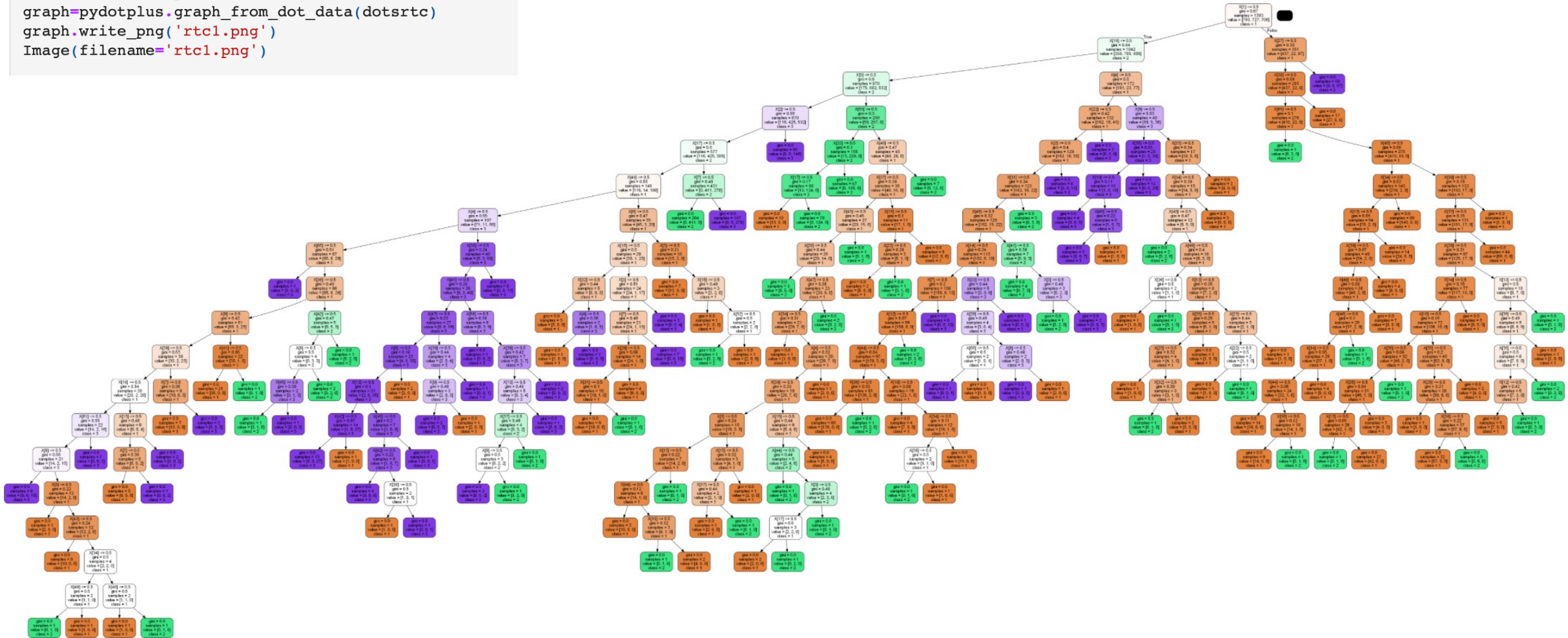
```
 VALIDACION RECALL:                precision    recall  f1-score   support

         1.0       0.97      1.00      0.99       201
         2.0       0.99      0.97      0.98       190
         3.0       1.00      0.99      1.00       166

    accuracy                           0.99       557
   macro avg       0.99      0.99      0.99       557
weighted avg       0.99      0.99      0.99       557

 PRUEBA RECALL:                  precision    recall  f1-score   support

         1.0       0.98      1.00      0.99       262
         2.0       0.99      0.97      0.98       197
         3.0       1.00      0.99      1.00       237

    accuracy                           0.99       696
   macro avg       0.99      0.99      0.99       696
weighted avg       0.99      0.99      0.99       696
```

```
clfdtc.feature_importances_
```

```
array([0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.4351717 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.40569868, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.11333185, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.04579777, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        ])
```

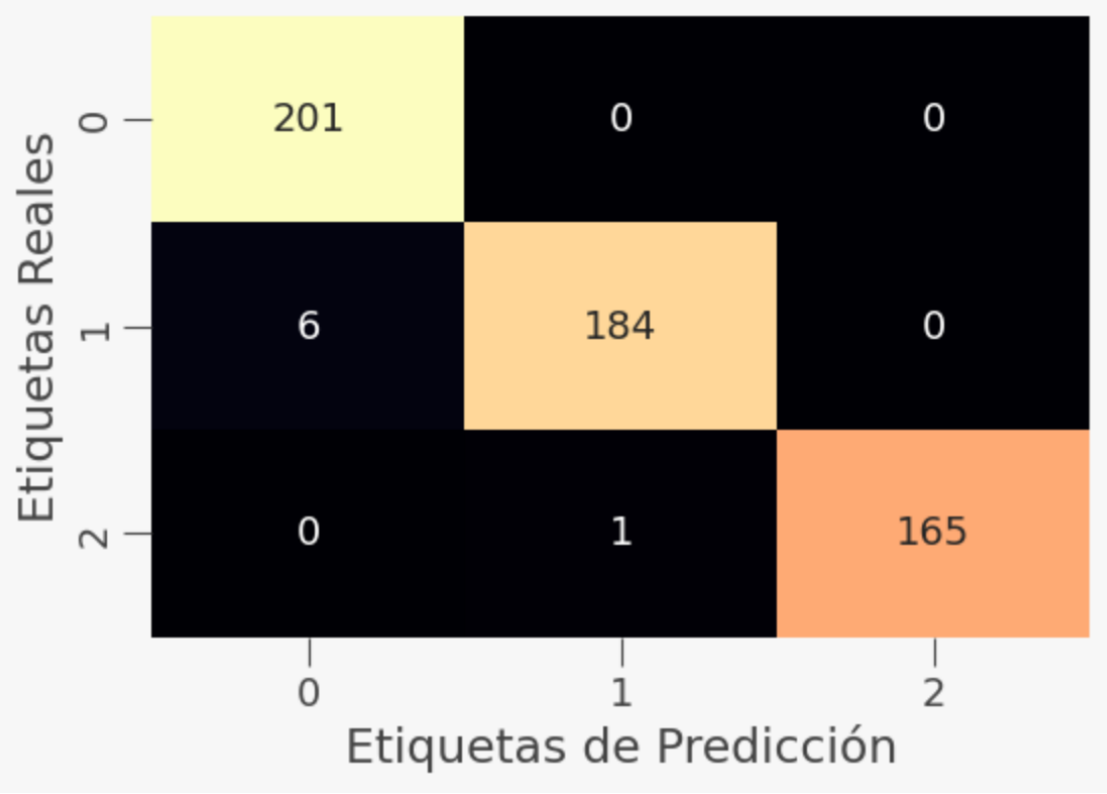
⋙Análisis

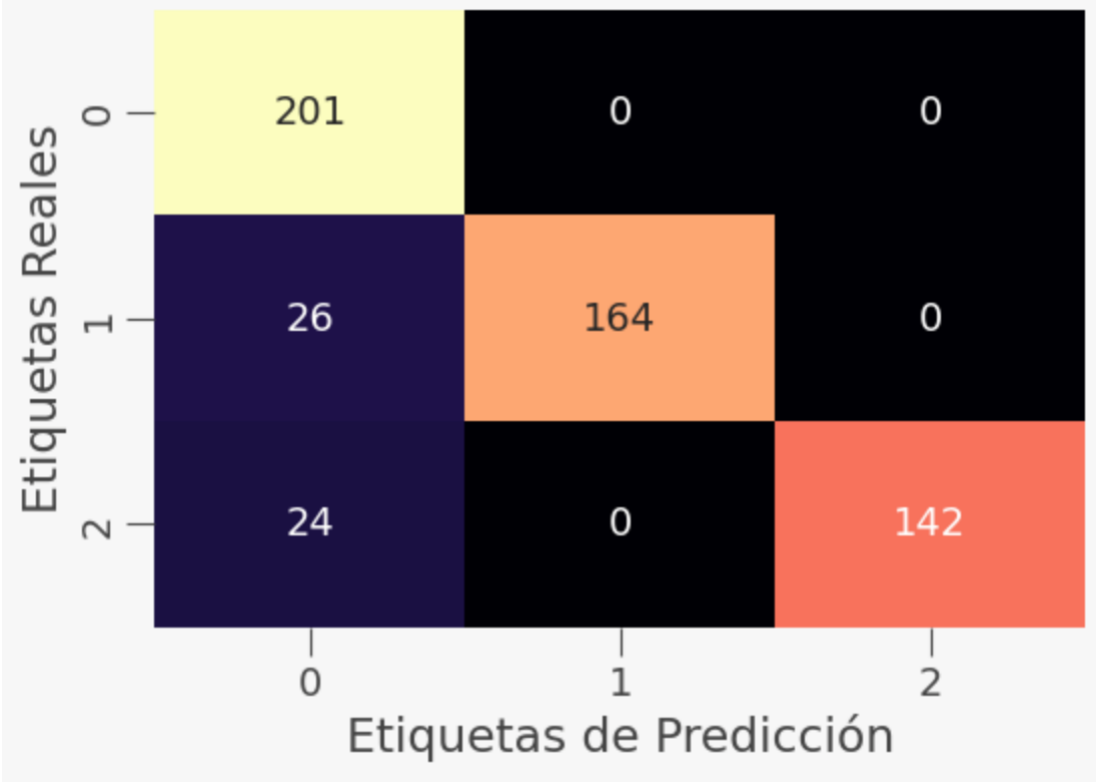
```
dotsrtc=export_graphviz(clfrtc.estimators_[0],
                class_names = ['1','2','3'],
                rounded = True, proportion = False,
                precision = 2, filled = True)
graph=pydotplus.graph_from_dot_data(dotsrtc)
graph.write_png('rtc1.png')
Image(filename='rtc1.png')
```



≫Confusion Matrix

```
# Confusion matrix

cm =confusion_matrix(y_val, y_hat_val)

plt.figure(figsize=(6,4))
ax = sns.heatmap(cm, annot=True,fmt='', cmap='magma', cbar=False)
ax.set(ylabel="Etiquetas Reales", xlabel="Etiquetas de Predicción")
plt.show()
```





≫Análisis

```
# RANDOM TREE MODEL BEST PARAMS
parameters = {
   'n_estimators': [5,10,15,25,50,100],
   'criterion':['gini', 'entropy', 'log_loss'],
   'max_depth':[1, 2, 3, 4, 5, 6, 8],
   'min_samples_split':[2,3,4,5],
   'max_leaf_nodes':[2,3,4,5],
   'class_weight':['balanced',None]
}

grid = GridSearchCV(
    estimator = rtc,
    param_grid=parameters,
    scoring='f1_weighted',
    n_jobs = -1,
    error_score='raise'
)
```

VALIDACION RECALL:

	precision	recall	f1-score	support
1.0	0.80	1.00	0.89	201
2.0	1.00	0.86	0.93	190
3.0	1.00	0.86	0.92	166
accuracy			0.91	557
macro avg	0.93	0.91	0.91	557
weighted avg	0.93	0.91	0.91	557

PRUEBA RECALL:

	precision	recall	f1-score	support
1.0	0.80	1.00	0.89	262
2.0	1.00	0.88	0.94	197
3.0	1.00	0.83	0.91	237
accuracy			0.91	696
macro avg	0.93	0.90	0.91	696
weighted avg	0.93	0.91	0.91	696

```
Mejor valor de exactitud obtenido con la mejor combinación: 0.9202483252246065
Mejor combinación de valores encontrados de los hiperparémetros: {'class_weight': None, 'criterion': 'entropy', 'max_depth': 4, 'max_leaf_
nodes': 5, 'min_samples_split': 4, 'n_estimators': 25}
Métrica utilizada: f1_weighted
```

≫Análisis

```python
# DESICION TREE MODEL BEST PARAMS
parameters = {
  'criterion':['gini', 'entropy', 'log_loss'],
  'max_depth':[1, 2, 3, 4, 5, 6, 8],
  'min_samples_split':[2,3,4,5],
  'max_leaf_nodes':[2,3,4,5],
  'class_weight':['balanced',None]
}

grid = GridSearchCV(
    estimator = dtc,
    param_grid=parameters,
    scoring='f1_weighted',
    n_jobs = -1,
    error_score='raise'
)

grid.fit(X_train, np.ravel(y_train))
```

VALIDACION RECALL:

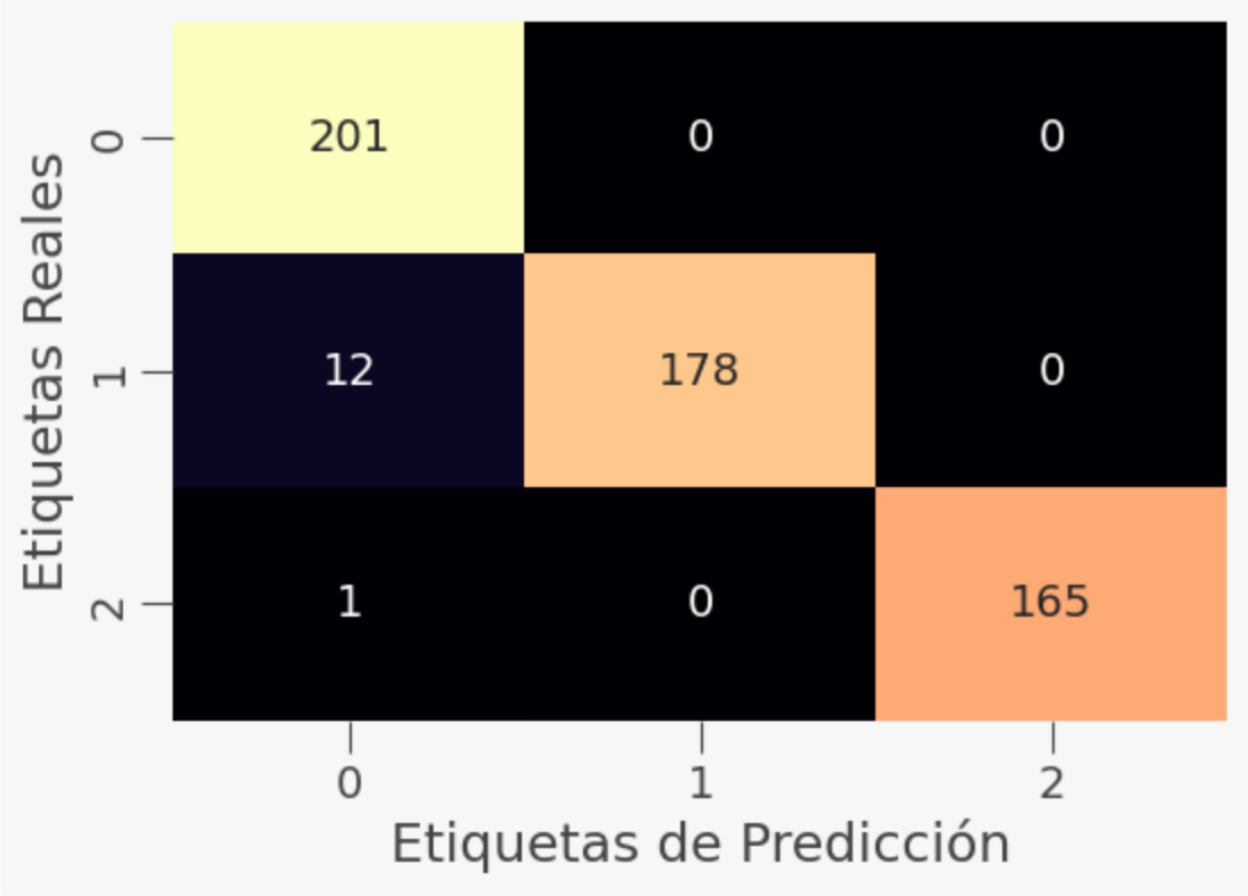
	precision	recall	f1-score	support
1.0	0.94	1.00	0.97	201
2.0	1.00	0.94	0.97	190
3.0	1.00	0.99	1.00	166
accuracy			0.98	557
macro avg	0.98	0.98	0.98	557
weighted avg	0.98	0.98	0.98	557

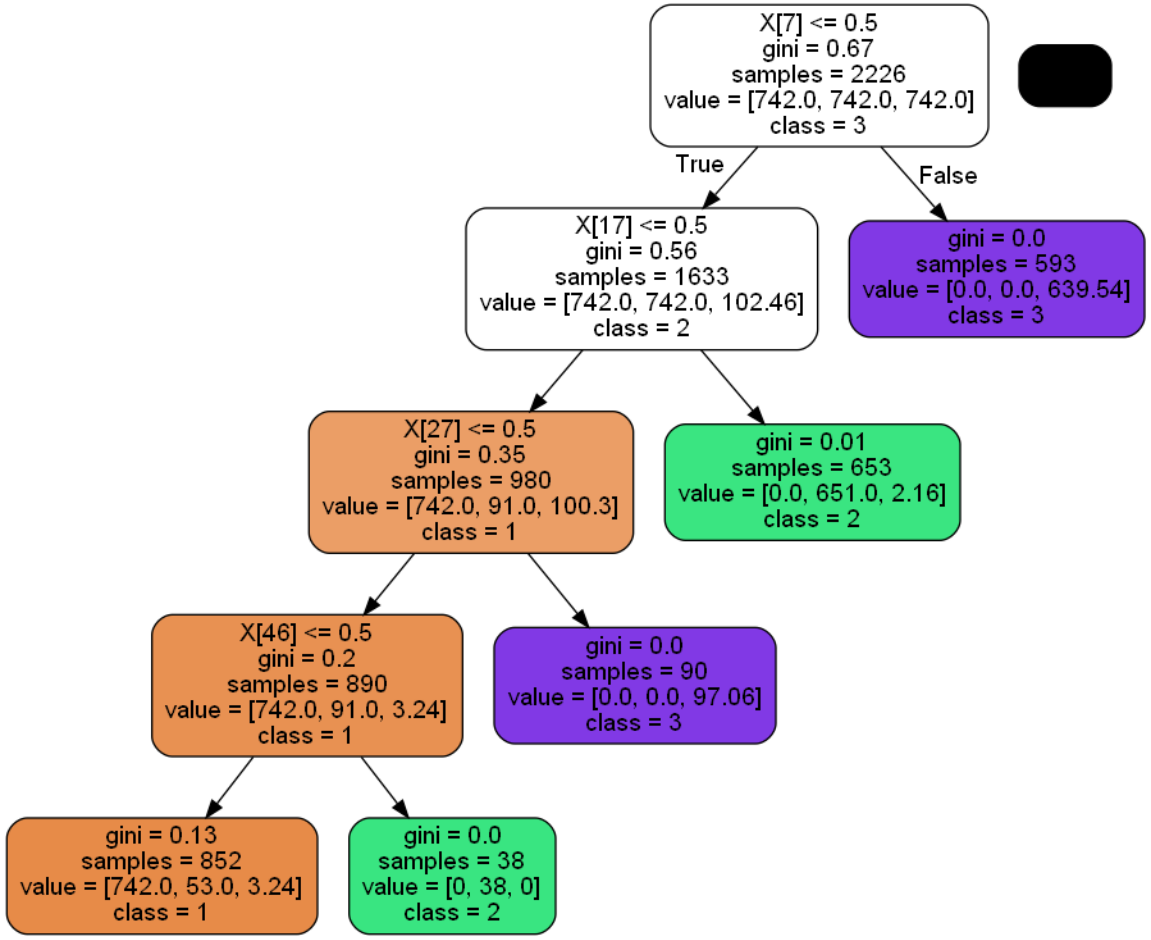
PRUEBA RECALL:

	precision	recall	f1-score	support
1.0	0.94	1.00	0.97	262
2.0	1.00	0.92	0.96	197
3.0	1.00	0.99	1.00	237
accuracy			0.98	696
macro avg	0.98	0.97	0.97	696
weighted avg	0.98	0.98	0.98	696

```
Mejor valor de exactitud obtenido con la mejor combinación: 0.9739026081929539
Mejor combinación de valores encontrados de los hiperparémetros: {'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 4, 'max_l
af_nodes': 5, 'min_samples_split': 2}
Métrica utilizada: f1_weighted
```

⋙Referencias





⋙Conclusiones

- Después de evaluar randomforest y decision tree, para el conjunto de datos de Agua superficiales; el mejor modelo de clasificación fue el de decision Tree, con base en los datos que obtuvimos .
- Las métricas fueron mucho mejor con decision Tree