UNIVERSIDAD NACIONAL DE COLOMBIA

Facultad de Minas
Departamento de Geociencias

Taller N°6

MÉTODOS ESTADÍSTICOS MULTIVARIADOS

Río Bedó, Mutatá-Antioquia

Carolina García Cadavid

Edier Aristizabal

Universidad Nacional de Colombia
Facultad de Minas
Cartografía Geotécnica

Noviembre 2023

Taller N°6

MÉTODOS ESTADÍSTICOS MULTIVARIADOS

Río Bedó, Mutatá-Antioquia

1. GENERALIDADES

La cuenca hidrográfica del Río Bedó, situada en la zona sur del municipio de Mutatá, en el departamento de Antioquia, Colombia. Esta cuenca se integra en la extensa red fluvial del Río Sucio, desempeñando un papel esencial como uno de sus afluentes.

En su cuenca alta y media, presenta grandes zonas muy boscosas. Entre los elementos que enriquecen la cuenca del Río Bedó, se destacan diversos afluentes que contribuyen a su caudal y carácter, entre ellos, se pueden mencionar el Río El Encanto, la Quebrada El Uvino, la Quebrada de Los Micos, la Quebrada La Bonga y la Quebrada Bedocito.

Además de su relevancia ambiental, la cuenca del Río Bedó también tiene importancia para las comunidades locales, que dependen de sus recursos hídricos y de los beneficios que brinda a la agricultura y la vida silvestre. Por tanto, es esencial garantizar su conservación y protección, no sólo como un ecosistema valioso, sino también como un activo fundamental para el bienestar de quienes habitan en su entorno.

- ÁREA → 40.93 km^2
- PERÍMETRO → 36.77 km
- ALTITUD MÁXIMA → 1350 msnm
- ALTITUD MÍNIMA → 155 msnm
- ALTURA PROMEDIO → 534 msnm
- LONG AXIAL LARGO → 11.5 km
- LONG AXIAL ANCHO → 7.05 km
- PENDIENTE PROMEDIO → 19.5°
- LONGITUD DEL CAUCE PRINCIPAL → Aprox. 11.14 km

2. Regresión logística

El método estadístico multivariado denominado regresión logística (RL) estima la relación de una variable dependiente categórica (la ocurrencia de movimientos en masa) de valores binarios de 0 (no ocurrencia) y 1 (si ocurrencia), con un grupo de variables independientes, en este caso las variables condicionantes del terreno.

La ventaja de la RL es que las variables predictoras no requieren tener distribución normal y pueden ser discretas o continuas, o una combinación de ambas.

A continuación se implementará el método de RL para evaluar la susceptibilidad por movimientos en masa. En este ejemplo sólo se utilizarán tres variables (pendiente, aspecto, curvatura y geología). Cabe aclarar que la variable elevación se correlaciona mucho con la variable geología a la hora de hacer la tabla con statismodels entonces daba un **error de colinealidad**, entonces tocaba eliminar una de las dos, por lo que **el modelo se hizo sin la variable elevación y se dejó la variable geología**.

Lo primero es que con nuestro DataFrame creado con anterioridad, se recomienda normalizar los valores. La RL es un algoritmo geométrico sensible a las unidades de cada variable.

```
[31]: d1={'inventario':inventario_vector_MenM,'pendiente':pendiente_vector_MenM,'aspecto':aspecto_vector_MenM, 'geologia':geologia_vector_MenM, 'curvatura':curvatura_vector_MenM}
      df1 = pd.DataFrame(d1)
      print(list(df1.columns))

      ['inventario', 'pendiente', 'aspecto', 'geologia', 'curvatura']
```

```
[32]: #Normalizamos las varibles de la data frame
      var_names2=['aspecto','pendiente', 'curvatura']
      for var in var_names2:
          df1[var]=(df1[var]-df1[var].mean())/df1[var].std()
      df1.head()
```

[32]:

	inventario	pendiente	aspecto	geologia	curvatura
0	0.0	-1.079078	-0.044614	3.0	-0.444951
1	0.0	-1.021525	-0.254136	3.0	-0.894841
2	0.0	-0.990910	0.124493	3.0	1.354610
3	0.0	-1.220003	-0.124896	3.0	0.454829
4	0.0	-1.367071	-0.044614	3.0	0.004939

Luego para resolver este modelo existen statsmodels y sklearn. La primera de ellas nos ofrece un resumen con los resultados y métricas de desempeño del modelo y utilizar la librería Patsy que permite el uso de fórmulas, esta fue la que se utilizó en este caso.

```
import statsmodels.formula.api as sfm
lr    = sfm.logit(formula = "inventario ~ pendiente + C(geologia) + aspecto + curvatura", data = df1).fit()
print(lr.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.118474
         Iterations: 35
C:\Users\carol\anaconda3\Lib\site-packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likel
  warnings.warn("Maximum Likelihood optimization failed to "
                           Logit Regression Results
==============================================================================
Dep. Variable:             inventario   No. Observations:               259233
Model:                          Logit   Df Residuals:                   259226
Method:                           MLE   Df Model:                            6
Date:                Tue, 21 Nov 2023   Pseudo R-squ.:                0.09257
Time:                        12:55:40   Log-Likelihood:                -30712.
converged:                      False   LL-Null:                       -33846.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================================
                           coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
Intercept               -2.6327      0.020   -131.016      0.000      -2.672      -2.593
C(geologia)[T.2.0]      -1.5294      0.026    -57.741      0.000      -1.581      -1.478
C(geologia)[T.3.0]     -47.9750   1.68e+09  -2.85e-08      1.000    -3.3e+09     3.3e+09
C(geologia)[T.15.0]    -15.4357    619.970     -0.025      0.980   -1230.554    1199.683
pendiente               -0.3382      0.014    -24.944      0.000      -0.365      -0.312
aspecto                  0.0745      0.011      6.618      0.000       0.052       0.097
curvatura               -0.0553      0.013     -4.187      0.000      -0.081      -0.029
================================================================================================
```

Entre los resultados se destacan el estimador para los coeficientes (máxima verosimilitud -MLE-), el logaritmo del estimador MLE (Log-likelihood), el coeficiente de ajuste (Pseudo R-squ.).

Con respecto a los coeficientes se presenta el valor de la prueba de hipótesis nula que el valor del coeficiente es igual a cero (z) como el valor del coeficiente entre el error estándar (std err), y el p-value (P>|z|), este valor debe ser menor al 5% (0.05) que en nuestro caso es de 0.0, lo cual significa que la probabilidad que el coeficiente tenga un valor de 0 es muy bajo. Finalmente se presenta el rango del 95% del dominio del valor del coeficiente.

Luego lo que hacemos en este método es parametrizar el modelo de diferentes formas, sin embargo requiere ingresar entre variables dependientes y variables independientes, haciendo la transformación la variable categórica así:

```
X=df1.drop("inventario", axis=1)
y=df1['inventario']
X.head()
```

	pendiente	aspecto	geologia	curvatura
0	-1.079078	-0.044614	3.0	-0.444951
1	-1.021525	-0.254136	3.0	-0.894841
2	-0.990910	0.124493	3.0	1.354610
3	-1.220003	-0.124896	3.0	0.454829
4	-1.367071	-0.044614	3.0	0.004939

```
dummy_geologia=pd.get_dummies(X['geologia'],prefix='geo')
column_name=X.columns.values.tolist()
column_name.remove('geologia')
X1=X[column_name].join(dummy_geologia)
X1.head()
```

	pendiente	aspecto	curvatura	geo_1.0	geo_2.0	geo_3.0	geo_15.0
0	-1.079078	-0.044614	-0.444951	0	0	1	0
1	-1.021525	-0.254136	-0.894841	0	0	1	0
2	-0.990910	0.124493	1.354610	0	0	1	0
3	-1.220003	-0.124896	0.454829	0	0	1	0
4	-1.367071	-0.044614	0.004939	0	0	1	0

Pasamos a importar esta función para RL, y se construye el modelo con los hiperparámetros. Estamos asignándole un class_weight = balanced para que asigne más peso a las celdas minoritarias, en este caso las celdas donde y=1. De esta forma RL nos ayuda a resolver el problema de la base de datos original desbalanceada. El segundo hiperparámetro que utilizamos es el tipo de algoritmo para resolver la RL, en este caso utilizaremos newton-cg. La librera "skilearn" utiliza por defecto el hiperparámetro fit_intercept con un valor de True que equivale a incluir el intercepto.

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression(class_weight='balanced', solver='newton-cg')
model
```

```
LogisticRegression
LogisticRegression(class_weight='balanced', solver='newton-cg')
```

Al tener el modelo construido, se le asignan los datos con la función fit, en este caso, primero la matriz con las variables predictoras, y luego la variable y. De esta forma el modelo se entrena con los datos y se pueden obtener los resultados de los valores de los coeficientes con la función coef_

```
result=model.fit(X1,y)
print(result.coef_)
```

```
[[-0.41004726 -0.01958664 -0.0871463   3.333273    1.8276798  -3.6375787
  -1.571927  ]]
```

Ya con nuestro modelo entrenado y ya guardado en la variable result podemos entonces obtener los valores que predice el modelo para toda la matriz con las variables predictoras. Para eso se utiliza la función predict. **Esta función clasifica cada celda como inestable (1) o estable (0).** Además, también se puede obtener los resultados del modelo antes de clasificarlo como (0,1). es decir la probabilidad de cada celda de ser 0 o de ser 1.

```
y_pred=result.predict(X1)
y_pred
```

```
array([0., 0., 0., ..., 1., 1., 1.])
```

```
y_prob=result.predict_proba(X1)
y_prob
```

```
array([[0.99648917, 0.00351084],
       [0.99641955, 0.00358048],
       [0.99711305, 0.00288693],
       ...,
       [0.20712996, 0.79287004],
       [0.18350607, 0.8164939 ],
       [0.20065725, 0.79934275]], dtype=float32)
```

```
y_probs=result.predict_proba(X1)[:,1]
y_probs
```

```
array([0.00351084, 0.00358048, 0.00288693, ..., 0.79287004, 0.8164939 ,
       0.79934275], dtype=float32)
```

Después lo que haremos es que construiremos entonces un nuevo DataFrame, pero en este caso con el vector completo, sin eliminar las celdas por fuera de la cuenca, y las celdas por fuera de la cuenca, en lugar de tener un valor de NaN les daremos un valor de 0.

```
pendiente_vector2=np.nan_to_num(pendiente_vector)
curvatura_vector2=np.nan_to_num(curvatura_vector)
aspecto_vector2=np.nan_to_num(aspecto_vector)
geologia_vector2=np.nan_to_num(geologia_vector)
```

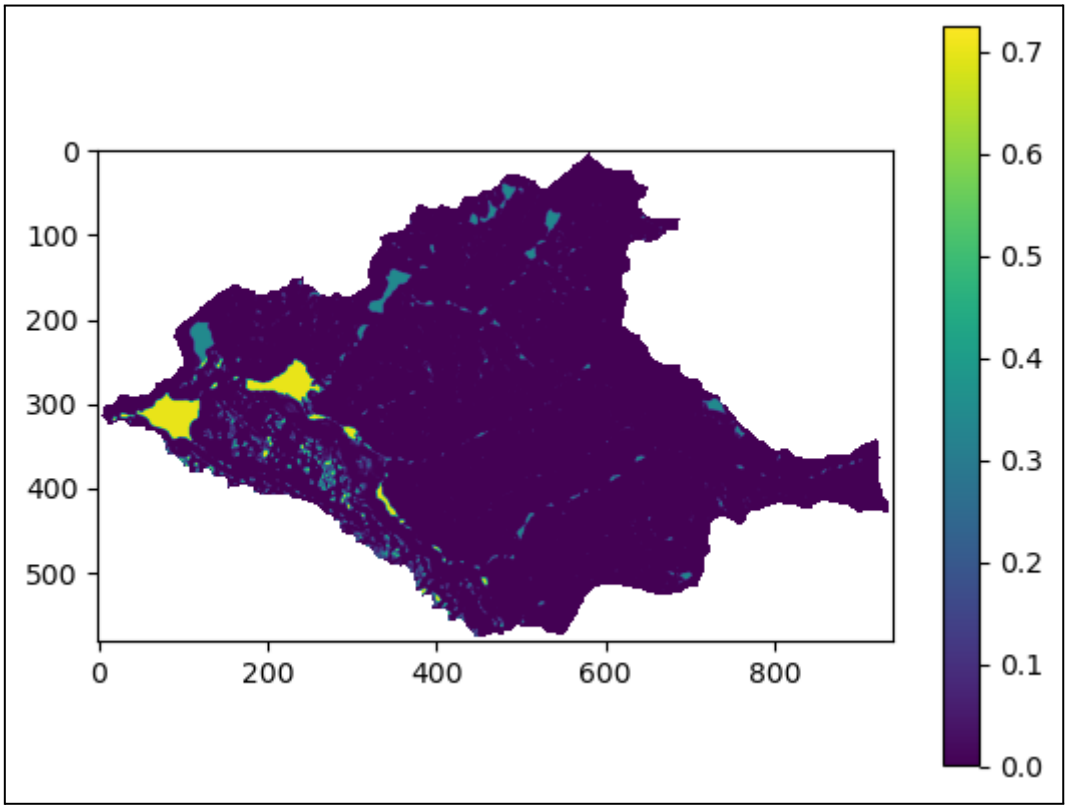
```
f={'pendiente':pendiente_vector2,'aspecto':aspecto_vector2,'geologia':geologia_vector2, 'curvatura':curvatura_vector2}
x_map=pd.DataFrame(f)
dummy_geologia=pd.get_dummies(x_map['geologia'],prefix='geo')
column_name=x_map.columns.values.tolist()
column_name.remove('geologia')
x_map=x_map[column_name].join(dummy_geologia)
x_map=x_map.drop('geo_0.0',axis=1)
```

Nos queda reconstruir a partir de dicho vector la matriz que conforma la cuenca con los valores de susceptibilidad. Para eso utilizaremos como máscara el mapa de la pendiente, dándonos el siguiente resultado del método de regresión logística para la cuenca del Río Bedó el Índice de Susceptibilidad.

```
y_pred=model.predict_proba(x_map)[:,1]
```

```
raster = rio.open('https://github.com/Caritos113/Cartograf-a-Geot-cnica/blob/main/Taller%203/Variables/Slope.tif?raw=true')
pendiente=raster.read(1)
```

```
IS=y_pred.reshape(pendiente.shape)
IS=np.where(pendiente<0,np.nan,IS)
plt.imshow(IS)
plt.colorbar();
```



3. Análisis discriminante lineal

El análisis discriminante lineal (LDA) es una técnica estadística que crea una función capaz de clasificar los fenómenos, teniendo en cuenta una serie de variables discriminadoras y una probabilidad de pertenencia.

Se importa la librería, se instancia el modelo con los hiperparámetros deseados, en este caso vamos a proyectar los datos sobre un nuevo eje (función discriminante), se entrena el modelo, y se predice, y luego se redimensiona este vector a la matriz que forma la cuenca, utilizando como máscara el mapa de pendiente. Pero para esto debemos construir un Dataframe con todas las celdas pero que en este caso sin la transformación de la variable geología. Estos modelos permiten trabajar con variables categóricas sin transformación.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

LDA = LinearDiscriminantAnalysis(n_components=1)
LDA
```

```
LinearDiscriminantAnalysis(n_components=1)
```

```
model = LDA.fit(X,y)
```

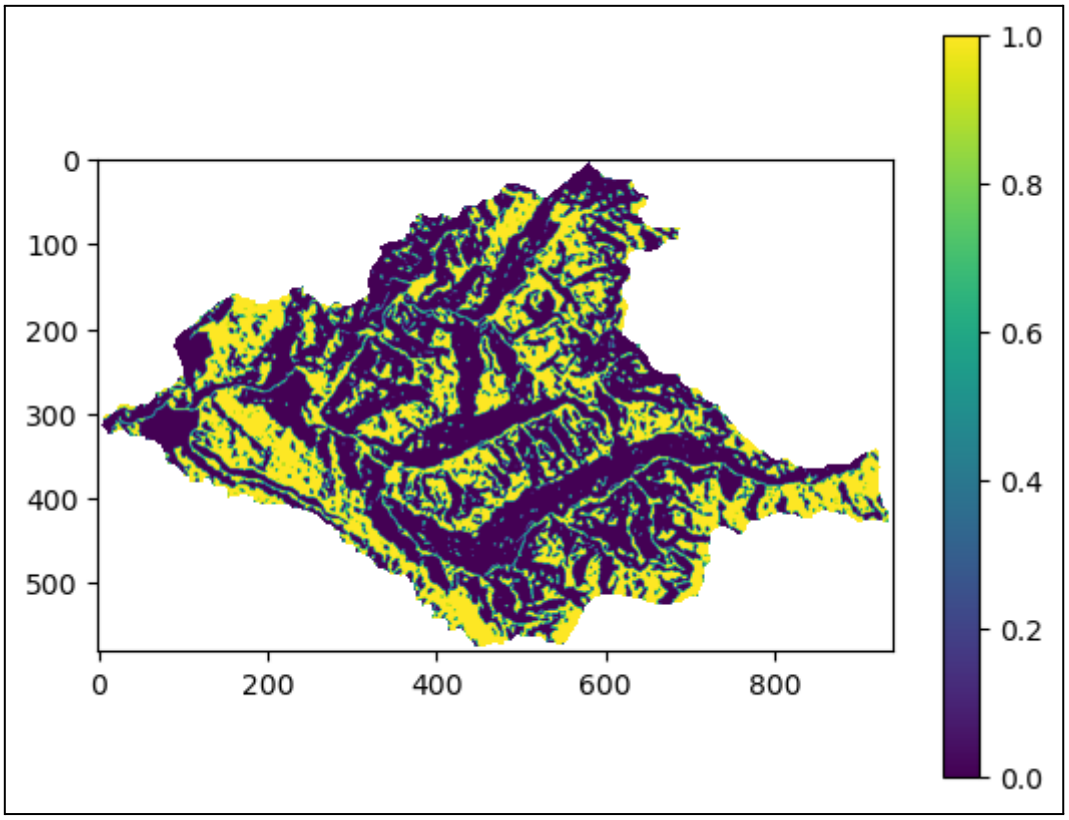
```
f={'pendiente':pendiente_vector2,'aspecto':aspecto_vector2,'geologia':geologia_vector2, 'curvatura':curvatura_vector2}
x_map=pd.DataFrame(f)
x_map
```

	pendiente	aspecto	geologia	curvatura
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0
...	...	...	...	...
547657	0.0	0.0	0.0	0.0
547658	0.0	0.0	0.0	0.0
547659	0.0	0.0	0.0	0.0
547660	0.0	0.0	0.0	0.0
547661	0.0	0.0	0.0	0.0

547662 rows × 4 columns

```
IS=model.predict(x_map)
```

```
IS=IS.reshape(pendiente.shape)
IS=np.where(pendiente<0,np.nan,IS)
plt.imshow(IS)
plt.colorbar();
```


0
100
200
300
400
500
0
200
400
600
800
1.0
0.8
0.6
0.4
0.2
0.0

4. Support vector machine

SVM es un modelo que representa a los puntos de muestra en el espacio, separando las clases a 2 espacios lo más amplios posibles mediante un hiperplano de separación definido como el vector entre los 2 puntos, de las 2 clases, más cercanos al que se llama vector soporte. En este caso se utilizará un kernel lineal y un valor de regularización de 1xE10. Existen otra serie de hiperparámetros para instanciar el modelo. Como este método no tiene un hiperparámetro para manejar esta situación, como RL, entonces debemos remuestrear nuestros datos de entrenamiento de tal forma que se reduzca el desbalance. En este caso utilizaremos el 1% de los no.

```
from sklearn.svm import SVC
model = SVC(kernel='linear', C=1E10)
```

```
df1=df0[(df0["inventario"]==1) | (df0["inventario"]==0).sample(frac=0.01)]
X_01=df1.drop("inventario", axis=1)
y_01=df1['inventario']
df1['inventario'].value_counts()
```

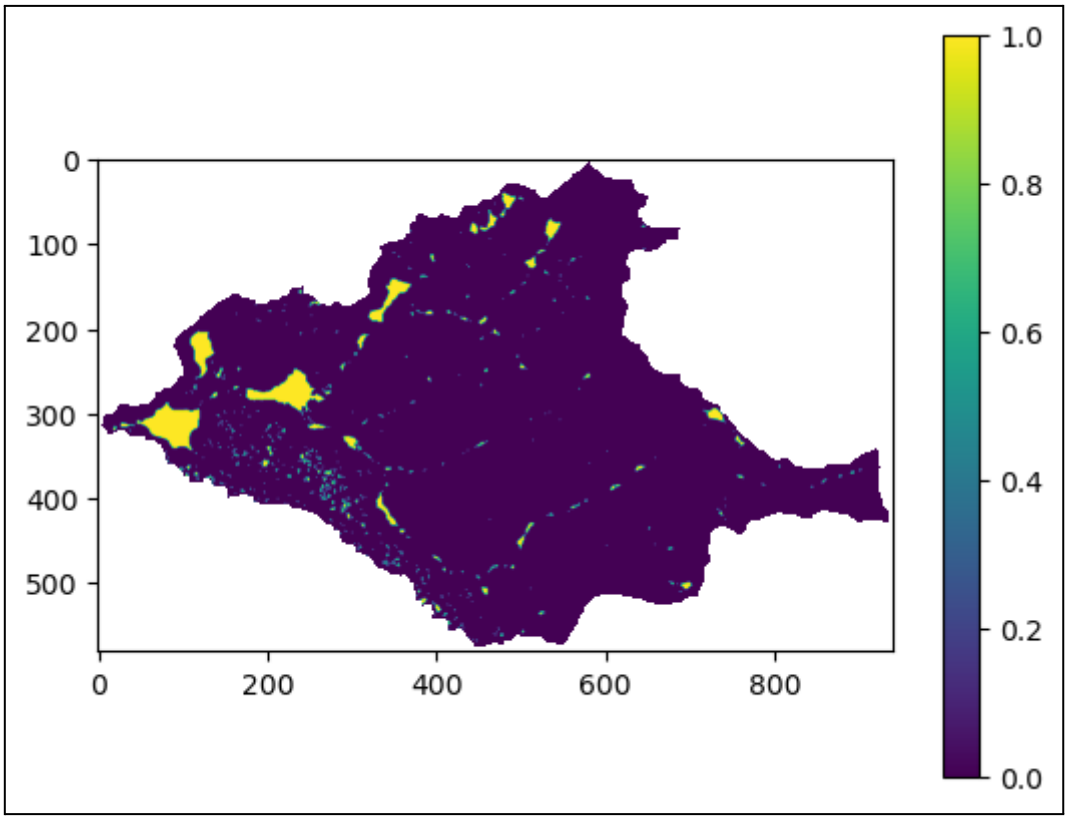
```
1.0    7467
0.0    2515
Name: inventario, dtype: int64
```

```
model.fit(X_01,y_01)
```

```
LinearDiscriminantAnalysis
LinearDiscriminantAnalysis(n_components=1)
```

```
IS_01 = model.predict(x_map)
```

```
IS_01=IS_01.reshape(pendiente.shape)
IS_01=np.where(pendiente<0,np.nan,IS_01)
plt.imshow(IS_01)
plt.colorbar();
```



5. Random forest

Un Random Forest es un conjunto de árboles de decisión combinados con bagging. Al usar bagging, lo que en realidad está pasando, es que distintos árboles ven distintas porciones de los datos. Ningún árbol ve todos los datos de entrenamiento. Esto hace que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, unos errores se compensan con otros y tenemos una predicción que generaliza mejor.

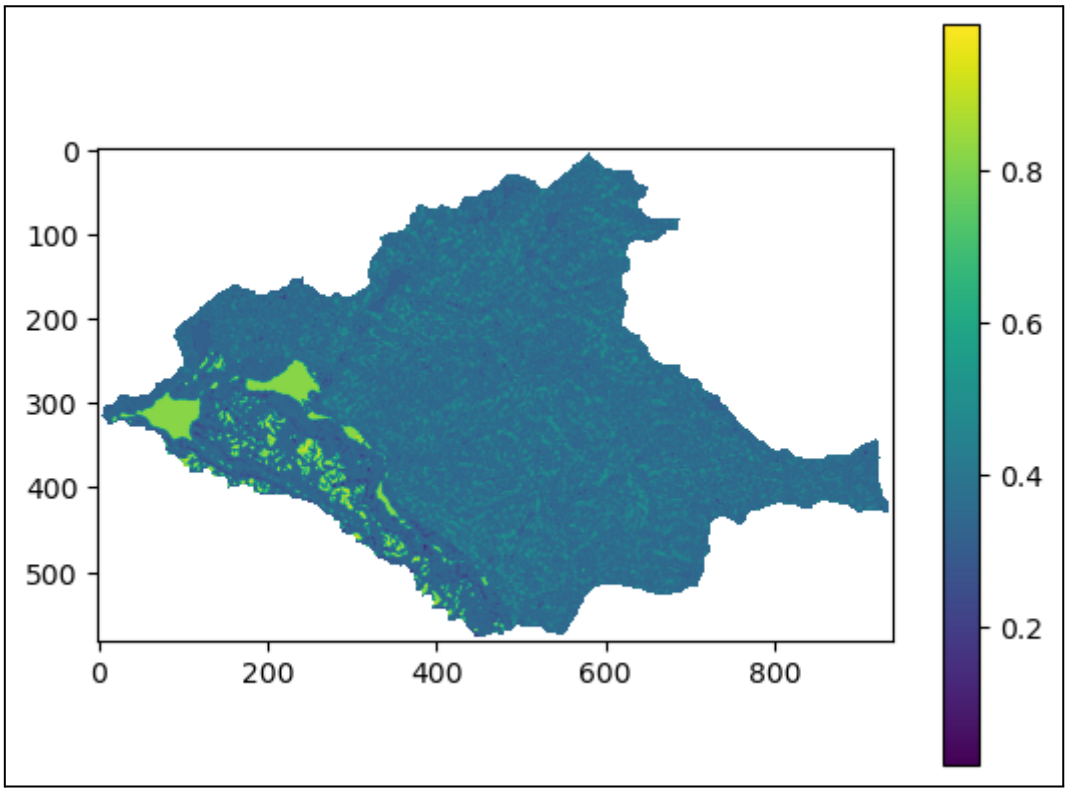
```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators= 1000)
```

```
rf.fit(X_01, y_01);
```

```
IS_01=rf.predict(x_map)
```

```
IS_01=IS_01.reshape(pendiente.shape)
IS_01=np.where(pendiente<0,np.nan,IS_01)
plt.imshow(IS_01)
plt.colorbar();
```



6. Redes neuronales

Una red neuronal artificial es un grupo interconectado de nodos similar a la vasta red de neuronas en un cerebro biológico. Cada nodo circular representa una neurona artificial y cada flecha representa una conexión desde la salida de una neurona a la entrada de otra.

Cada neurona está conectada con otras a través de unos enlaces. En estos enlaces el valor de salida de la neurona anterior es multiplicado por un valor de peso. Estos pesos en los enlaces pueden incrementar o inhibir el estado de activación de las neuronas adyacentes.

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(5,2),max_iter=500)  # dos capas escondidas, la primera con 5 neuronas y la segunda con dos neuronas
mlp
```

```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_01)
X_trans = scaler.transform(X_01)
```

```
mlp.fit(X_01,y_01)
```

```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500)
```

```
IS=mlp.predict(x_map)
```

```
IS_01=IS_01.reshape(pendiente.shape)
IS_01=np.where(pendiente<0,np.nan,IS_01)
plt.imshow(IS_01)
plt.colorbar();
```