Entrega Final Proyecto IA “Forest Cover Type Prediction”

Bryan Zuleta Vélez
Sol Yajhaira Linares Mateus
Daniel Alejandro Yepes Mesa

Facultad de Ingeniería
Universidad de Antioquia

Introducción a la inteligencia artificial para ciencias e ingeniería

Profesor: Raul Ramos Pollan

12 de Noviembre de 2022

UNIVERSIDAD DE ANTIOQUIA
1 8 0 3

Forest Cover Type Prediction

1. INTRODUCCIÓN

La cobertura forestal es crucial para la salud del suelo, el ciclo del agua, el clima y la calidad del aire, de ahí la importancia de analizar cuál es la cobertura forestal adecuada para cada bosque; en el caso de este proyecto, analizaremos esto para los bosques que se encuentran en la región 2 del servicio forestal de estados unidos, este análisis se realiza teniendo en cuenta principalmente la función de sus suelos, y otros factores.

Para este proyecto se va a generar una predicción de la cobertura forestal (basándonos en la categoría donde predomina la cubierta arbórea) de un bosque a partir de variables estrictamente cartográficas. Los datos reales se obtienen a partir de los datos del Sistema de Información de Recursos de la Región 2 del Servicio Forestal de EE.UU (USFS).

Los resultados posibles para esta clasificación son:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

2. EXPLORACIÓN DESCRIPTIVA DEL DATASET

Para iniciar con el procesamiento de los datos vamos a usar tres archivos *.csv*; el primer archivo será ***Train.csv***, donde obtendremos información como:

1. Elevation - Elevación en metros
2. Aspect - Aspecto en grados azimut
3. Slope – Pendiente en grados
4. Horizontal_Distance_To_Hydrology - Horz distancia a las características de agua superficial más cercanas
5. Vertical_Distance_To_Hydrology - Vert distancia a las características de agua superficial más cercanas
6. Horizontal_Distance_To_Roadways - Horz Dist a la carretera más cercana
7. Hillshade_9am (0 to 255 index) - Índice de sombreado a las 9 a.m., solsticio de verano
8. Hillshade_Noon (0 to 255 index) - Índice de sombreado al mediodía, solsticio de verano
9. Hillshade_3pm (0 to 255 index) - Índice de sombreado a las 15:00, solsticio de verano Horizontal_Distance_To_Fire_Points - Horz Dist a los puntos de ignición de incendios forestales más cercanos
10. Wilderness_Area (4 binary columns, 0 = absence or 1 = presence) - Designación de área silvestre
11. Soil_Type (40 binary columns, 0 = absence or 1 = presence) - Designación del tipo de suelo
12. Cover_Type (7 types, integers 1 to 7) - Designación de tipo de cubierta forestal

El segundo archivo que es ***Test.csv****,* a parte de la información contenida en train.csv, nos dará los valores de la columna *Cover_Type* y por último el tercer archivo es **sampleSubmission** que contiene las observaciones aplicadas al archivo *Test.csv*, con el cual se obtiene el resultado de acierto del modelo desarrollado.

Para analizar el comportamiento de los datos y realizar los respectivos modelos se procedió a ubicar en una carpeta de drive los archivos test y train, para luego crear un acceso directo al colab respectivo para iniciar con el procesamiento de este.

Luego de dicho procesado se analizó el comportamiento de cada variable de manera independiente, verificando la existencia de datos atípicos o fuera de lo establecido, en este proceso notamos que nuestro Dataset no contenía datos faltantes, así que usamos un método para generar un número fijo de datos nulos.

```
df_forest = df.copy()

from random import randrange

porcentaje = 6

n_filas, n_columnas = df.shape
numero_nans = (n_filas*n_columnas*porcentaje)//100  # // es la división entera

for i in range(numero_nans):
  fila = randrange(0, n_filas)
  columna = randrange(1, n_columnas-1)
  df_forest.iloc[fila, columna] = float("nan")
```

```
df_forest.head(20)
```

	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3p
0	1	2596.0	51.0	3.0	258.0	0.0	510.0	NaN	232.0	148
1	2	2590.0	56.0	2.0	212.0	-6.0	390.0	220.0	235.0	151
2	3	2804.0	139.0	NaN	268.0	65.0	3180.0	234.0	238.0	Na
3	4	2785.0	NaN	18.0	242.0	118.0	3090.0	238.0	238.0	122
4	5	2595.0	45.0	2.0	153.0	-1.0	391.0	220.0	234.0	150
5	6	2579.0	NaN	6.0	300.0	-15.0	67.0	230.0	237.0	140
6	7	2606.0	45.0	7.0	270.0	5.0	633.0	NaN	225.0	138
7	8	2605.0	49.0	4.0	234.0	7.0	573.0	222.0	230.0	144
8	9	2617.0	45.0	9.0	240.0	56.0	666.0	223.0	221.0	133
9	10	2612.0	NaN	10.0	247.0	11.0	636.0	228.0	219.0	124
10	11	2612.0	201.0	4.0	180.0	51.0	735.0	218.0	243.0	161
11	12	2886.0	151.0	11.0	371.0	26.0	NaN	234.0	240.0	136

Debido al tamaño de nuestro dataset, también a que solo tenemos una columna fija (ID) y a la combinación entre números enteros y booleanos en las columnas que teníamos que rellenar para el entrenamiento, se crearon dos ciclos *for*, uno es para aquellos datos que son números enteros que se llenó con el promedio de los valores y el otro para las columnas con valores booleanos que se llenó con valores de 1's y 0's.

```
aux = 0

for column in columns_means:
  df_forest[column].fillna(value=means_vector[aux], inplace=True)
  aux += 1


for column in columns_bool:
  df_forest[column].fillna(value=random.randint(0,1), inplace=True)
```

Verificamos que los valores NaN fueron cambiados segun cada caso

```
df_forest.head(100)
```

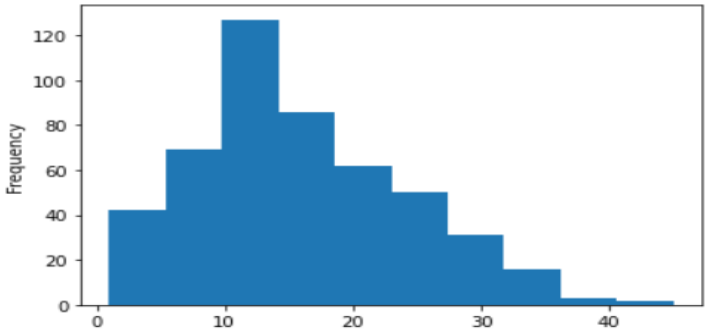
	Id	Elevation	Aspect	Slope	Horizontal_Distance_To_Hydrology	Vertical_Distance_To_Hydrology	Horizontal_Distance_To_Roadways	Hillshade_9am	Hillshade_Noon	Hillshade_3
0	1	2596.0	51.0	3.0	258.0	0.0	510.0	213.7	232.0	14
1	2	2590.0	56.0	2.0	212.0	-6.0	390.0	220.0	235.0	15
2	3	2804.0	139.0	15.9	268.0	65.0	3180.0	234.0	238.0	13
3	4	2785.0	154.7	18.0	242.0	118.0	3090.0	238.0	238.0	12
4	5	2595.0	45.0	2.0	153.0	-1.0	391.0	220.0	234.0	15
...	...	...	...	...	...	...	...	...	...	
95	96	2860.0	31.0	10.0	295.0	98.0	3644.0	213.7	218.0	13
96	97	3067.0	164.0	11.0	85.0	7.0	6811.0	230.0	243.0	14
97	98	2804.0	72.0	5.0	543.0	61.0	3115.0	225.0	231.0	13
98	99	2562.0	59.0	15.9	0.0	0.0	1116.0	221.0	233.0	14
99	100	2775.4	34.0	9.0	190.0	16.0	1790.9	213.7	221.0	13

100 rows × 56 columns

A pesar del tamaño de la base de datos, no tuvimos problemas a la hora de procesar los datos, así que realizamos un entrenamiento para ver su funcionamiento y revisar si estaba adaptada a lo que se nos solicitaba.

Para analizar los datos se generó un histograma que mostraba en grupos de 5 la frecuencia de los datos, además usamos una función para relacionar los datos de la columna Cover_Type, y el resultado fue:

```
data_forest['Slope'].plot(kind = "hist")
```



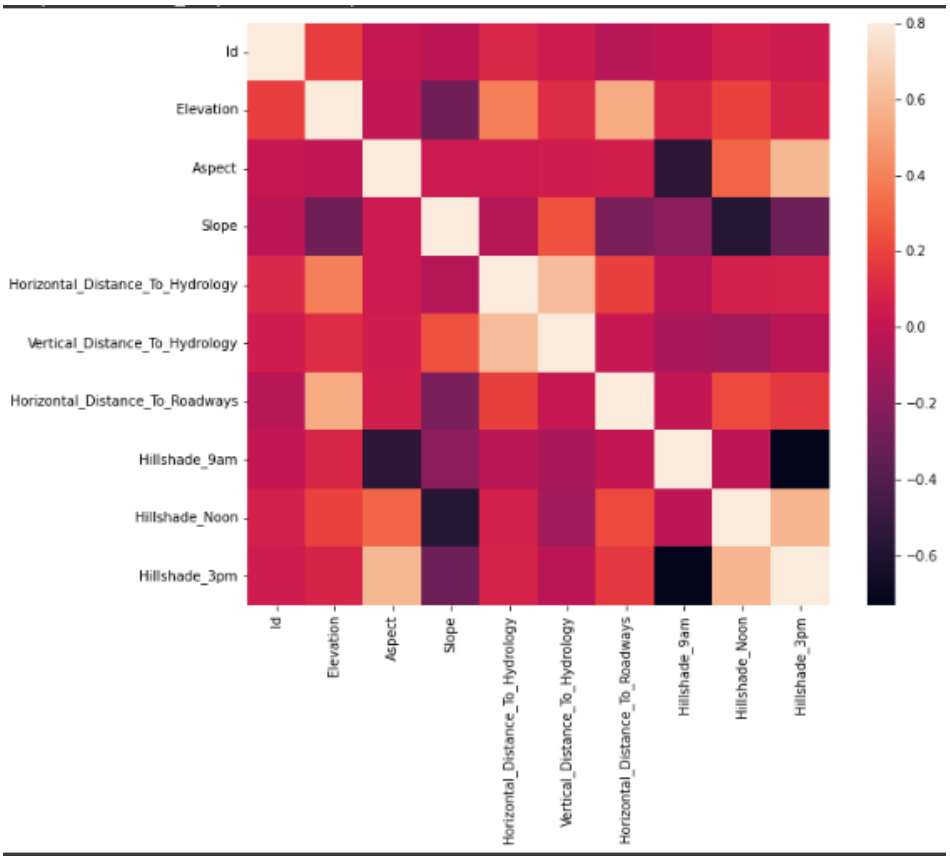
c) Vemos la relacion de las columnas con respecto a el Cover_Type

```
abs(df_forest.corr()['Cover_Type'].sort_values(ascending=False))
```

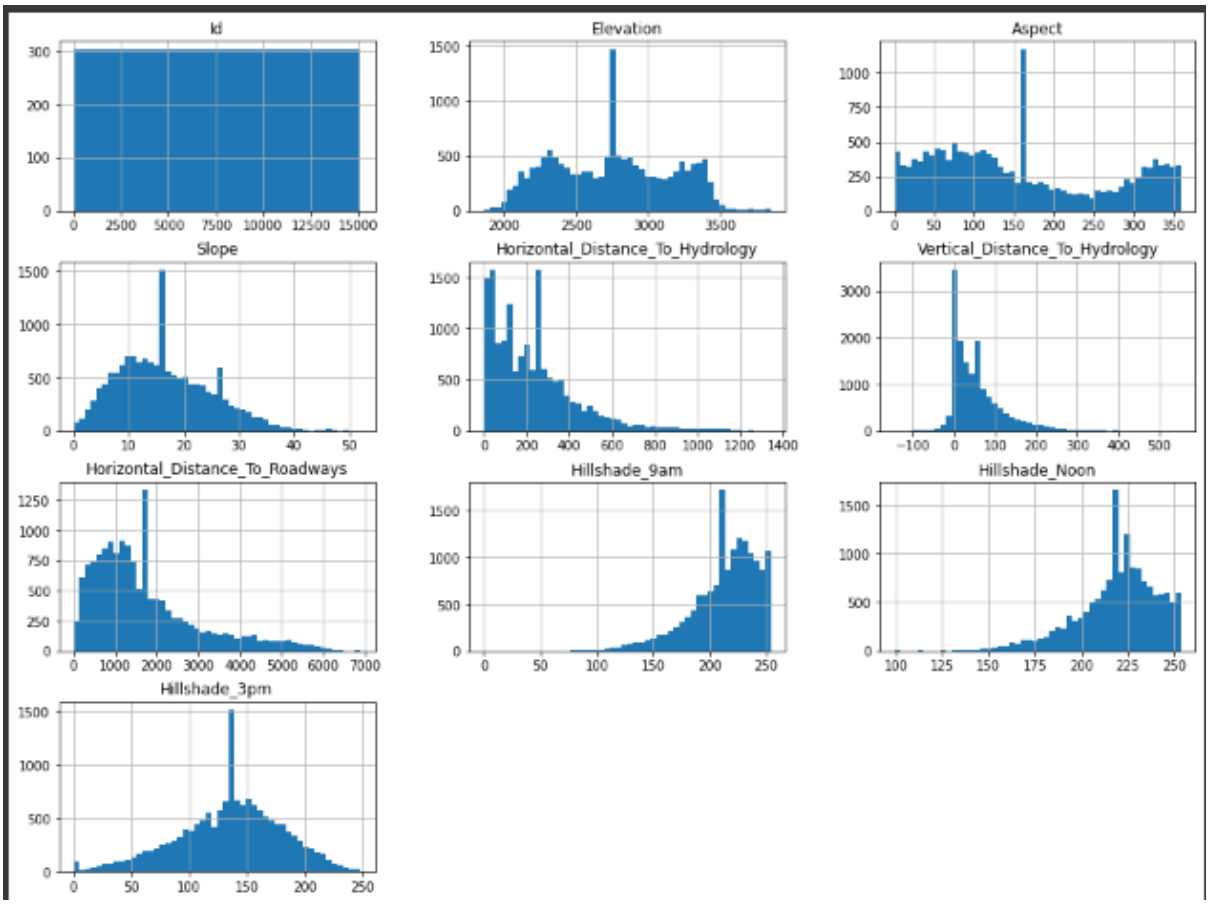
```
Cover_Type                          1.000000
Soil_Type40                         0.202362
Soil_Type38                         0.171100
Soil_Type39                         0.154101
Soil_Type10                         0.124331
Wilderness_Area3                    0.116512
Id                                  0.108363
Slope                               0.085433
Wilderness_Area4                    0.072565
Vertical_Distance_To_Hydrology      0.071291
Soil_Type37                         0.071210
Soil_Type13                         0.029777
Soil_Type35                         0.025531
Soil_Type17                         0.021722
Soil_Type2                          0.021125
Soil_Type14                         0.020748
Wilderness_Area2                    0.016303
```

en donde se puede notar que el tipo de suelo 38, 39, 40, 10 y 35 son los que más están relacionados con la cobertura que se está buscando, aunque tiene relación con todos, los mencionados son los que más tienen fuerza con referencia al Cover_Type.

Aquí mostramos una representación de un mapa de calor de la matriz de correlación que hay entre las variables y cual se relaciona más una con la otra



La media de la característica varía de 16 a 2749. STD para Horizontal_Distance_To_Roadways que es el dato más disperso, seguido de Horizontal_Distance_To_Fire_Points y Elevation. La más definida y cercana a la media es SLOPE seguida de las 3 características de HillSHADE. Todas las entidades tienen un valor mínimo de 0 excepto las entidades Elevation y Vertical_Distance_To_Hydrology. Elevation tiene el valor mínimo más alto y Vertical_Distance_To_Hydrology tiene un valor negativo. Las entidades de HILLSHADE, excepto Hillshade_3pm, tienen un valor máximo similar. Horizontal_Distance_To_Fire_Points tiene el valor máximo más alto seguido de las entidades Horizontal_Distance_To_Roadways. También tienen los rangos más altos de todas las características. SLOPE tiene el valor máximo y el rango más bajos. La función Aspect sigue de cerca este mismo concepto. Es bueno tener en cuenta que la razón por la que algunas características están muy extendidas y tienen valores altos es porque 5 de las 10 variables se miden en metros. Estas variables son: Elevation,Horizontal_Distance_To_Hydrology,Vertical_Distance_To_Hydrology,Horizontal_Distance_To_Roadways,Horizontal_Distance_To_Fire_Points. Esto tiene sentido que estos tengan valores y rangos altos. Las funciones como ASPECT y SLOPE se miden en grados, lo que significa que los valores máximos no pueden superar los 360. Las funciones de HILLSHADE solo pueden tomar un valor máximo de 255.


Id
Elevation
Aspect
Slope
Horizontal_Distance_To_Hydrology
Vertical_Distance_To_Hydrology
Horizontal_Distance_To_Roadways
Hillshade_9am
Hillshade_Noon
Hillshade_3pm

3. ITERACIONES DE DESARROLLO

Para realizar predicciones relacionadas con el comportamiento futuro de los datos, es decir conocer las diferentes coberturas de suelo y tipos de suelo, se realizaron múltiples modelos como:

A. Método GaussianoNB

```
def classification_gs(df_forest,y):

    x=df_forest
    y=y
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)


    predict=GaussianNB().fit(x_train,y_train).predict(x_test)

    accuracy = accuracy_score(y_test,predict)
    precision = precision_score(y_test,predict,average='micro')
    recall = recall_score(y_test,predict,average='micro')
    f1 = f1_score(y_test,predict,average='micro')

    return accuracy, precision, recall, f1

classification_gs(x,y)
```

```
(0.5691137566137566,
 0.5691137566137566,
 0.5691137566137566,
 0.5691137566137566)
```

B. Método Bernoulli

```
def classification_bn(df_forest,y):


    x=df_forest
    y=y
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)


    predict=BernoulliNB().fit(x_train,y_train).predict(x_test)

    accuracy = accuracy_score(y_test,predict)
    precision = precision_score(y_test,predict,average='micro')
    recall = recall_score(y_test,predict,average='micro')
    f1 = f1_score(y_test,predict,average='micro')

    return accuracy, precision, recall, f1

classification_bn(x,y)
```

```
(0.5575396825396826,
 0.5575396825396826,
 0.5575396825396826,
 0.5575396825396826)
```

C. DecisionTree

Metodo DecisionTree

```
def classification_dTree(df_forest,y):


    x=df_forest
    y=y
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)


    predict=DecisionTreeClassifier().fit(x_train,y_train).predict(x_test)

    accuracy = accuracy_score(y_test,predict)
    precision = precision_score(y_test,predict,average='micro')
    recall = recall_score(y_test,predict,average='micro')
    f1 = f1_score(y_test,predict,average='micro')

    return accuracy, precision, recall, f1

classification_dTree(x,y)
```

```
(0.7255291005291006,
 0.7255291005291006,
 0.7255291005291006,
 0.7255291005291006)
```

D. GradientBoosting

Metodo de GradientBoosting

```
def classification_graBoosting(df_forest,y):


    x=df_forest
    y=y
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)

    predict=GradientBoostingClassifier().fit(x_train,y_train).predict(x_test)

    accuracy = accuracy_score(y_test,predict)
    precision = precision_score(y_test,predict,average='micro')
    recall = recall_score(y_test,predict,average='micro')
    f1 = f1_score(y_test,predict,average='micro')

    return accuracy, precision, recall, f1

classification_graBoosting(x,y)
```

```
(0.7668650793650794,
 0.7668650793650794,
 0.7668650793650794,
 0.7668650793650794)
```

E. RandomForest

metodo de RandomForest

```
def classification_ramForest(df_forest,y):


    x=df_forest
    y=y
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random_state=42)

    predict=RandomForestClassifier().fit(x_train,y_train).predict(x_test)

    accuracy = accuracy_score(y_test,predict)
    precision = precision_score(y_test,predict,average='micro')
    recall = recall_score(y_test,predict,average='micro')
    f1 = f1_score(y_test,predict,average='micro')

    return accuracy, precision, recall, f1

classification_ramForest(x,y)
```

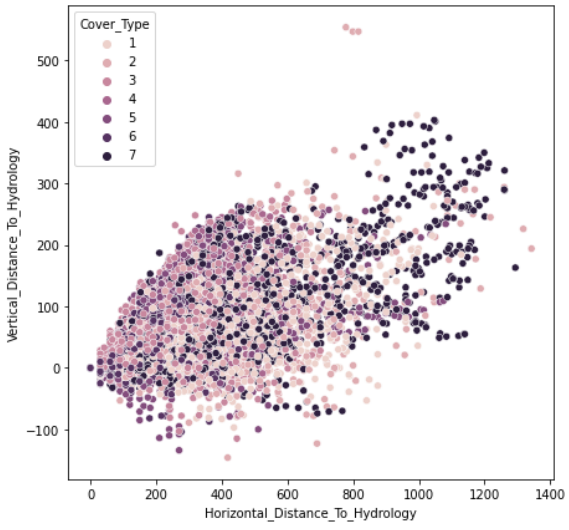
```
(0.8095238095238095,
 0.8095238095238095,
 0.8095238095238095,
 0.8095238095238095)
```

F. Clustering

```
[24] import matplotlib.pyplot as plt
     import seaborn as sns

     plt.figure(figsize = (7,7))
     sns.scatterplot("Horizontal_Distance_To_Hydrology","Vertical_Distance_To_Hydrology",hue = "Cover_Type",data = df_forest)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f629a9db2d0>
```



Una vez se realizaron todos los métodos anteriormente enunciados pudimos notar que algunos nos entregaban mayor precisión que otros por lo cual seleccionamos los 3 que tienen mayor exactitud.

Con el fin de dar un mejor análisis creamos un *dataframe* con los datos que se obtienen de estos tres modelo

```
top3Metods = pd.DataFrame(columns=['Acurracy', 'Precision', 'Recall', 'F1'],index=['DecisionTree','GradientBoosting','RandomForest'])
dTree = classification_dTree(x,y)
top3Metods.Acurracy[0] = dTree[0]
top3Metods.Precision[0] = dTree[1]
top3Metods.Recall[0] = dTree[2]
top3Metods.F1[0] = dTree[3]

graBoosting = classification_graBoosting(x,y)
top3Metods.Acurracy[1] = graBoosting[0]
top3Metods.Precision[1] = graBoosting[1]
top3Metods.Recall[1] = graBoosting[2]
top3Metods.F1[1] = graBoosting[3]

RamForest = classification_ramForest(x,y)
top3Metods.Acurracy[2] = RamForest[0]
top3Metods.Precision[2] = RamForest[1]
top3Metods.Recall[2] = RamForest[2]
top3Metods.F1[2] = RamForest[3]


top3Metods.head()
```

	Acurracy	Precision	Recall	F1
DecisionTree	0.731481	0.731481	0.731481	0.731481
GradientBoosting	0.766865	0.766865	0.766865	0.766865
RandomForest	0.814815	0.814815	0.814815	0.814815

Podemos ver en la tabla que el método que nos entrega la mayor exactitud con las diferentes métricas es el RandomForest. Por lo cual para el entrenamiento usaremos este modelo.

La métrica empleada será multi-class classification accuracy, definimos la exactitud (accuracy en inglés) como el ratio entre las predicciones correctas (suma de verdaderos positivos y verdaderos negativos) y las predicciones totales. Scikit-Learn implementa la métrica sklearn.metrics.accuracy_score que puede utilizarse en clasificación binomial y multiclase y que devuelve el porcentaje de predicciones correctas. Lógicamente, el clasificador ideal tendría una exactitud de 1 (todas las muestras serían bien clasificadas) y el peor clasificador posible tendría una exactitud de 0 (ninguna muestra sería bien clasificada).

Teniendo esto claro, procedemos a analizar el modelo RandomForest, y realizar el entrenamiento, con los datos obtenidos del procesamiento del dataset.

```
predict=model.predict(df_test.drop(labels=['Id'],axis=1))
```

```
submission= pd.DataFrame(data = predict,columns = ['Cover_Type'])
submission.head(100)
```

	Cover_Type
0	2
1	2
2	2
3	2
4	2
...	...
95	2
96	2
97	2
98	2
99	2

100 rows × 1 columns

Podemos observar en la tabla que aparentemente todos los resultados dan para un Cover_Type 2, pero si analizamos mas afondo el data frame podemos ver que aunque el tipo 2 es el que mas predomina, existen otros tipos de suelos.

Por último, generamos un archivo .csv con los resultados. y los almacenamos en el proyecto

```
submission['Id'] =df_test["Id"]
submission.set_index('Id',inplace=True)
submission.to_csv('submission.csv')
```

4. RETOS Y CONSIDERACIONES DE DESPLIEGUE

1.El reto principal que tuvimos a la hora de hacer nuestro proyecto es que los datos que íbamos a manejar no cumplían con uno de los requisitos que se habían dado en el transcurso de la materia que era tener un % de datos faltantes, cosa que nuestro dataset no tenía porque era un dataset con todos los datos completos, entonces fue necesario generar un código para poder llenar éste porcentaje.

2. Otro de los retos que se nos generó fue el conocimiento de modelos supervisados y no supervisados, tuvimos que investigar hasta cierto punto sobre los modelos no supervisados y llevarlos a implementar en nuestro proyecto.

3. También generar la matriz de correlación se pudo considerar un pequeño reto ya qué tuvimos que generar una relación de factores para poder entender parte del proyecto.

4. La interpretación de los resultados de los modelos y llevarlos al contexto de lo que es el problema.

5. CONCLUSIONES

1. En todo el desarrollo del proyecto pudimos darnos cuenta que el tipo de cubierta forestal en cada zona estaba muy correlación nada con la distancia a las fuentes hídricas esto afecta demasiado el resultado final a la hora de tomar decisión con mayor exactitud
2. Pudimos observar que el cover type 2 es el que predomina mayormente, puesto que este como se muestra en el clustering es quien se encuentra más presente en los datos de entrenamiento además su distribución con respecto a las diferentes distancias de las fuente hidrológicas es la mejor localizada
3. Pudimos notar que por algunos modelos matemáticos aunque entregaban unas métricas interesantes sus tiempos de cómputo eran demasiado extensos lo cual lo podría hacer inviable si este modelo se quería llevar a zonas más grandes del país, o incluso a el país entero, esto podría ser un problema a futuro
4. Según los datos expuestos en nuestro proyecto, podemos observar que entre todos los modelos que manejamos, el que se acerca con mayor exactitud es el Random Forest ya que manejó predicciones bastante elevadas con respecto al resto, gracias a esto logramos llegar a nuestro objetivo.
5. Se puede concluir que el tipo más usual de cobertura es la tipo 2 sin embargo también aparecen de manera frecuente coberturas tipo 5 o 4, entre otras. esto nos permite estar seguros que no existe alguna inconsistencia en el entrenamiento
6. Podemos determinar gracias a este modelo y a el entrenamiento de esta IA que con casi un 86% de exactitud la utilización de esta para la toma de decisiones puede ser acertada, a no ser que factores externos no considerados en el data frame entren a jugar algún papel primario o secundario