Tecnológico Nacional de México

Instituto Tecnológico De Hermosillo

Proyecto Final

Gonzalez Varela Eli Ronaldo

Maestro: Eduardo Antonio Hinojosa Palafox

Grupo: S9A

Hermosillo, Sonora **12 de Diciembre de 2022**

Introducción

El auge de la ciencia de datos y por ende campos como el machine learning e inteligencia artificial ha permitido dar propuestas de soluciones a problemas en las que se involucren reconocimiento de patrones y detección de tendencias. La minería de datos y el aprendizaje automático han permitido el desarrollo de soluciones. En estos campos los objetos de estudio se describen mediante vectores multidisciplinarios representados por un conjunto de variables conocidos como atributos, rasgos o características). En este tipo de problemas un humano por lo general define las variables que potencialmente son útiles para caracterizar o representar un conjunto de datos, sin embargo existe la posibilidad de que existan variables irrelevantes o redundantes a la contribución de las tareas de clasificación o análisis de datos. El propósito de la minería y aprendizaje automático de datos es el de computarizar y automatizar estos procesos de selección de características con el fin de dar predicciones y clasificaciones a partir de un conjunto de datos dado. Esto ha permitido un análisis más profundo de tendencias y patrones que los procesos realizados de forma humana permitiendo tener un rol más importante en el mundo moderno.

El siguiente documento busca la elaboración de una clasificación de un conjunto de datos de COVID-19 con el fin de determinar si un paciente está o no contagiado con la enfermedad. El COVID-19 sigue siendo uno de los temas más explorados y analizados por el impacto que tuvo en los últimos años y sigue siendo hasta la fecha por lo que presenta una oportunidad para demostrar la funcionalidad y aplicabilidad de los modelos de clasificación en una problemática actual.

Descripción del problema a desarrollar

La pandemia de COVID-19 tuvo un impacto a nivel mundial ocasionando múltiples cierres, cancelaciones y repercusiones negativas en diferentes secciones y disciplinas. Algo único de la pandemia fue que sucedió durante la era de la ciencia de datos y la inteligencia artificial por lo que el enfoque de muchos gobiernos fue de guiarse por políticas guiadas por la ciencia y enfocarse en datos y no fechas. Por ende los científicos, clínicos y aquellos en el área de la ciencia de datos e inteligencia artificial se han tomado la tarea de trabajar en el descubrimiento de información con el fin de tomar acción e impactar en las vidas diarias. Como científicos de datos es importante observar la situación de diferentes perspectivas y buscar la mejor solución e interpretación de datos con el fin de dar propuesta a futuras situaciones similares y actuales. El problema que se cubrirá específicamente del COVID-19 es la detección de características en pacientes para determinar cuales influyen en la infección y tomar acciones preventivas.

Descripción del conjunto de datos

Este conjunto de datos contiene una enorme cantidad de información anonimizada relacionada con los pacientes, incluidas las condiciones previas. El conjunto de datos en bruto consta de 21 características únicas y 1.048.576 pacientes únicos. En las características booleanas, 1 significa "sí" y 2 significa "no". Los valores 97 y 99 son datos que faltan.

sexo: femenino o masculino

edad: del paciente.

clasificación: resultados de la prueba covid. Los valores 1-3 significan que al paciente se le diagnosticó covídica en diferentes grados. 4 o superior significa que el paciente no es portador de covid o que la prueba no es concluyente.

tipo de paciente: hospitalizado o no hospitalizado.

neumonía: si el paciente ya tiene inflamación de los sacos aéreos o no.

embarazo: si la paciente está embarazada o no.

diabetes: si el paciente tiene diabetes o no.

EPOC: indica si el paciente padece o no enfermedad pulmonar obstructiva crónica.

asma: si el paciente tiene asma o no.

inmsupr: si el paciente está inmunodeprimido o no.

hipertensión: si el paciente tiene hipertensión o no.

cardiovascular: si el paciente tiene una enfermedad relacionada con el corazón o los vasos sanguíneos.

renal crónica: si el paciente tiene enfermedad renal crónica o no.

otra enfermedad: si el paciente padece o no otra enfermedad.

obesidad: si el paciente es obeso o no.

tabaco: si el paciente es consumidor de tabaco.

usmr: indica si el paciente trató unidades médicas de primer, segundo o tercer nivel.

unidad médica: tipo de institución del Sistema Nacional de Salud que proporcionó la atención.

intubado: indica si el paciente estaba conectado al ventilador.

UCI: indica si el paciente había sido ingresado en una Unidad de Cuidados Intensivos.

muerte: indica si el paciente falleció o se recuperó.

Descripción de la solución propuesta.

La solución propuesta para este problema es el de la utilización de métodos de ensamblaje y selección de hiper parámetros para el modelo con mejor rendimiento utilizando GridSearch. Los métodos de ensamblaje se han revelado como un potente método para mejorar la solidez y la precisión de las soluciones supervisadas y no supervisadas. Además, como continuamente se generan enormes cantidades de datos desde distintos puntos de vista, es importante consolidar distintos conceptos para una toma de decisiones inteligentes. GridSearch, por otro lado, permite la ejecución de múltiples iteraciones de modelos con el fin de seleccionar los parámetros que mejor encajan en la clasificación deseada. Es importante además denotar los atributos que afectan más en la clasificación y predicción de contagio por lo que también se utilizará un método para determinar cuales características influyen más en dicho procesamiento. Además se realizará un análisis exploratorio de los datos permitiendo observar relaciones entre los diferentes atributos además de limpiar dichos datos en caso de alguna inconsistencia.

Descripción del código utilizado

```
[ ]  from google.colab import drive

     drive.mount('/content/drive')
```

Se comienza importando el dataset subido anteriormente a Google Drive debido al tamaño del dataset.

```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
```

Luego se importan las librerías correspondientes para el trabajo con datos.

```
df = pd.read_csv('/content/drive/MyDrive/Covid Data.csv')
```

Se crea un dataframe leyendo el archivo separado por comas utilizando el método de la librería Pandas.

```
[ ]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 21 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   USMER                 1048575 non-null  int64
 1   MEDICAL_UNIT          1048575 non-null  int64
 2   SEX                   1048575 non-null  int64
 3   PATIENT_TYPE          1048575 non-null  int64
 4   DATE_DIED             1048575 non-null  object
 5   INTUBED               1048575 non-null  int64
 6   PNEUMONIA             1048575 non-null  int64
 7   AGE                   1048575 non-null  int64
 8   PREGNANT              1048575 non-null  int64
 9   DIABETES              1048575 non-null  int64
 10  COPD                  1048575 non-null  int64
 11  ASTHMA                1048575 non-null  int64
 12  INMSUPR               1048575 non-null  int64
 13  HIPERTENSION          1048575 non-null  int64
 14  OTHER_DISEASE         1048575 non-null  int64
 15  CARDIOVASCULAR        1048575 non-null  int64
 16  OBESITY               1048575 non-null  int64
 17  RENAL_CHRONIC         1048575 non-null  int64
 18  TOBACCO               1048575 non-null  int64
 19  CLASIFFICATION_FINAL  1048575 non-null  int64
 20  ICU                   1048575 non-null  int64
dtypes: int64(20), object(1)
memory usage: 168.0+ MB
```

Las columnas o atributos disponibles del dataset son las siguientes y cada una representa diferentes condiciones de los pacientes denotadas por 1 y 2 para valores mientras que 97 y 98 se utilizan para valores nulos. La fecha es la única columna que no tiene valores binarios.

```
[ ] df.head()
```

	USMER	MEDICAL_UNIT	SEX	PATIENT_TYPE	DATE_DIED	INTUBED	PNEUMONIA	AGE	PREGNANT	D
0	2	1	1	1	03/05/2020	97	1	65	2	
1	2	1	2	1	03/06/2020	97	1	72	97	
2	2	1	2	2	09/06/2020	1	2	55	97	
3	2	1	1	1	12/06/2020	97	2	53	2	
4	2	1	2	1	21/06/2020	97	2	68	97	

5 rows × 21 columns

Con el método head se obtienen los primeros cinco registros del dataset y se observa que hay valores con 97 por lo que será necesario cambiarlos.

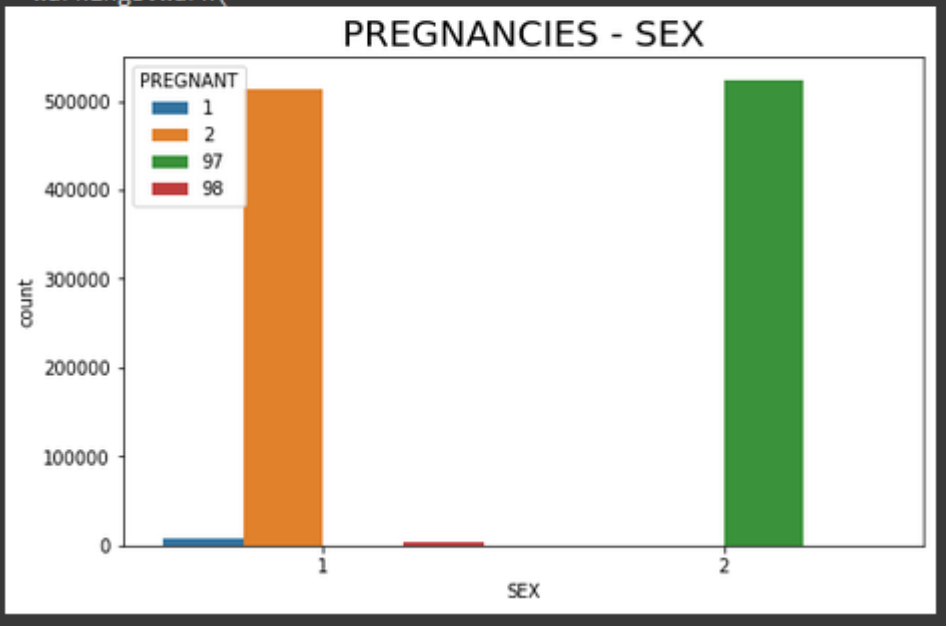
```
df.isnull().sum()

USMER                0
MEDICAL_UNIT         0
SEX                  0
PATIENT_TYPE         0
DATE_DIED            0
INTUBED              0
PNEUMONIA            0
AGE                  0
PREGNANT             0
DIABETES             0
COPD                 0
ASTHMA               0
INMSUPR              0
HIPERTENSION         0
OTHER_DISEASE        0
CARDIOVASCULAR       0
OBESITY              0
RENAL_CHRONIC        0
TOBACCO              0
CLASIFFICATION_FINAL 0
ICU                  0
dtype: int64
```

Con *isnull().sum()* se buscan valores nulos con el fin de verificar si existen otros valores que no concuerden.

Análisis Exploratorio de Datos

```
[ ]  plt.figure(figsize = (8,5))
     ax = sns.countplot(df.SEX, hue = df.PREGNANT)
     plt.title("PREGNANCIES - SEX", fontsize = 20);
```

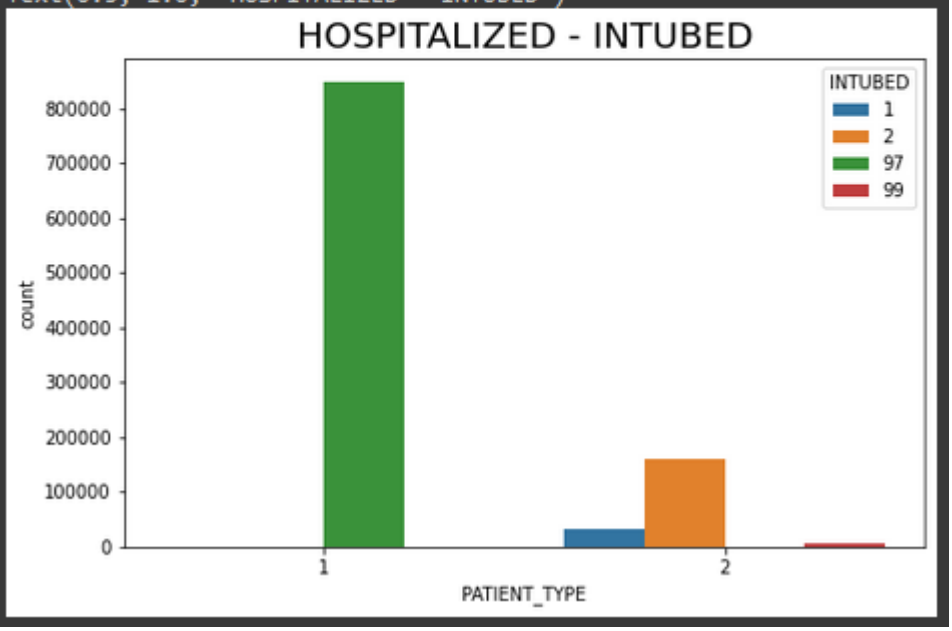
```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: F
  warnings.warn(
```



Se hizo luego un análisis exploratorio en el que se observan los valores de las diferentes columnas y filas del dataset. El primero analizado es el de embarazos y sexo. Algo notable es que en la columna de embarazos 98 significa nulo para mujeres y 97 para hombres. Para eso será necesario cambiar de 97 a 2 porque los hombres no pueden estar embarazados.

```
[ ]  df.PREGNANT = df.PREGNANT.replace(98, 2)
     df.PREGNANT = df.PREGNANT.replace(97, 2)
```

```
[10] plt.figure(figsize = (8,5))
     ax = sns.countplot(df.PATIENT_TYPE, hue = df.INTUBED)
     plt.title("HOSPITALIZED - INTUBED", fontsize = 20)

     /usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: F
       warnings.warn(
     Text(0.5, 1.0, 'HOSPITALIZED - INTUBED')
```

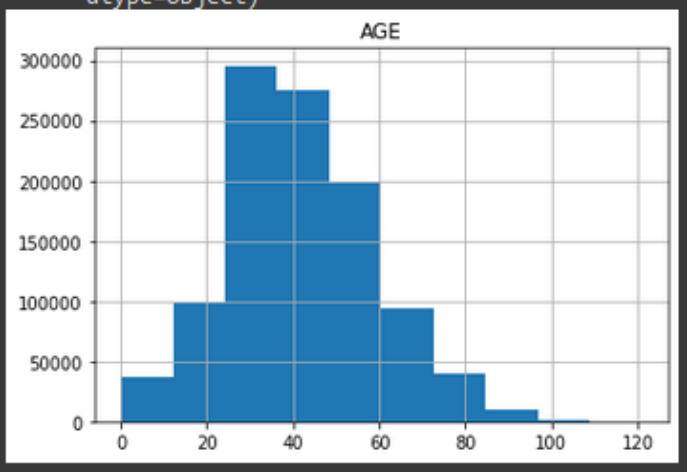


La siguiente relación es la de hospitalizados con pacientes intubados. Existen muchos valores con 97 por lo que es mejor borrar la columna.

```
[5]  df.drop("INTUBED", axis = 1, inplace = True)
```

```
df.hist(column='AGE')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f456563faf0>]],
      dtype=object)
```



Mediante un histograma de edades se observan los rangos de edad más comunes. En general se tienen entre los 20 y los 60 años.

Los valores de la variable clasificadora tienen alrededor de 5 valores adicionales no deseados. La descripción del dataset nos indica que los valores 1, 2, 3 son positivos a COVID mientras que 4, 5, 6, 7 son negativos a COVID. Por lo que se convertirá a una clasificación binaria.a

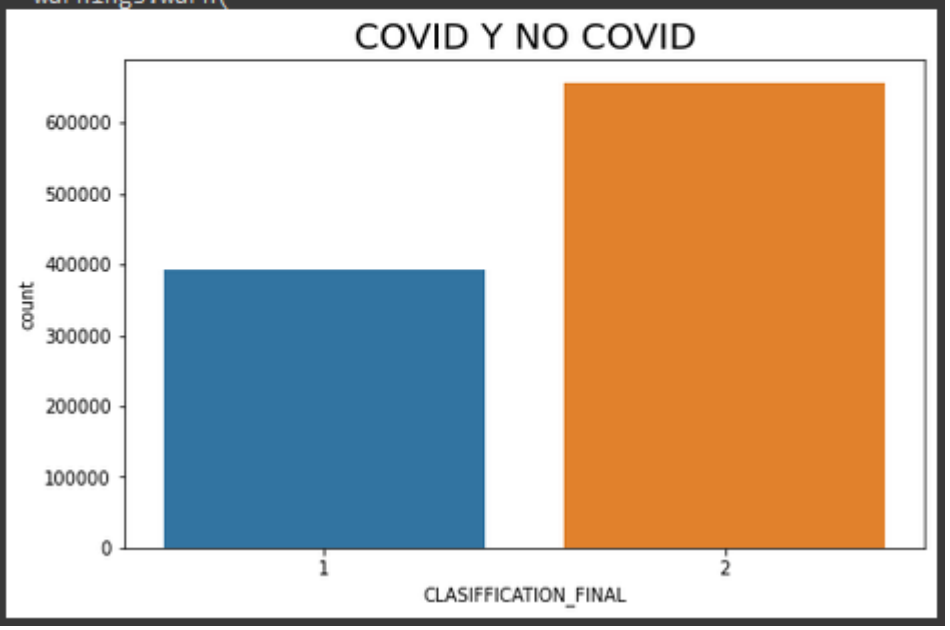
```
[7]  df.CLASIFFICATION_FINAL.value_counts()

     7    499250
     3    381527
     6    128133
     5     26091
     1      8601
     4      3122
     2      1851
     Name: CLASIFFICATION_FINAL, dtype: int64
```

```
[8]  df.CLASIFFICATION_FINAL = df.CLASIFFICATION_FINAL.replace([1,2,3], 1)
     df.CLASIFFICATION_FINAL = df.CLASIFFICATION_FINAL.replace([4,5,6,7], 2)
```

Sería ideal hacer un análisis de las muertes. Para ello se analizará primero todos aquellos que salieron positivo y negativo a la enfermedad

```
plt.figure(figsize = (8,5))
ax = sns.countplot(df.CLASIFFICATION_FINAL)
plt.title("COVID Y NO COVID", fontsize = 20);
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: F
  warnings.warn(
```



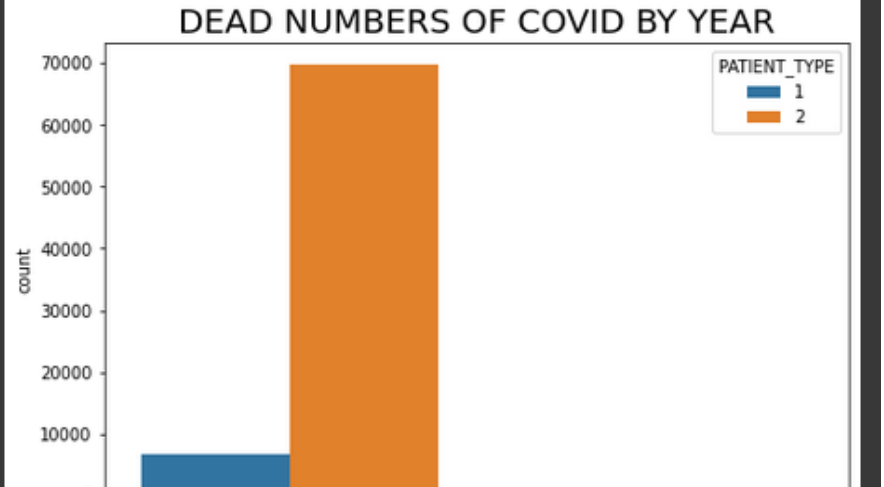
Hay una diferencia de alrededor de 200,000 personas entre los negativos y positivos, pero aún así la cantidad de positivos es alta.

Luego se puede separar la fecha en años y meses para observar diferentes picos y tiempos.

```
df["MONTH"] = df.DATE_DIED.str.split("/", expand = True)[1]
df["YEAR"] = df.DATE_DIED.str.split("/", expand = True)[2]
```

```
plt.figure(figsize = (8,5))
ax = sns.countplot(x = "YEAR", hue = "PATIENT_TYPE", data = df)
plt.title("DEAD NUMBERS OF COVID BY YEAR", fontsize = 20)
```

```
Text(0.5, 1.0, 'DEAD NUMBERS OF COVID BY YEAR')
```

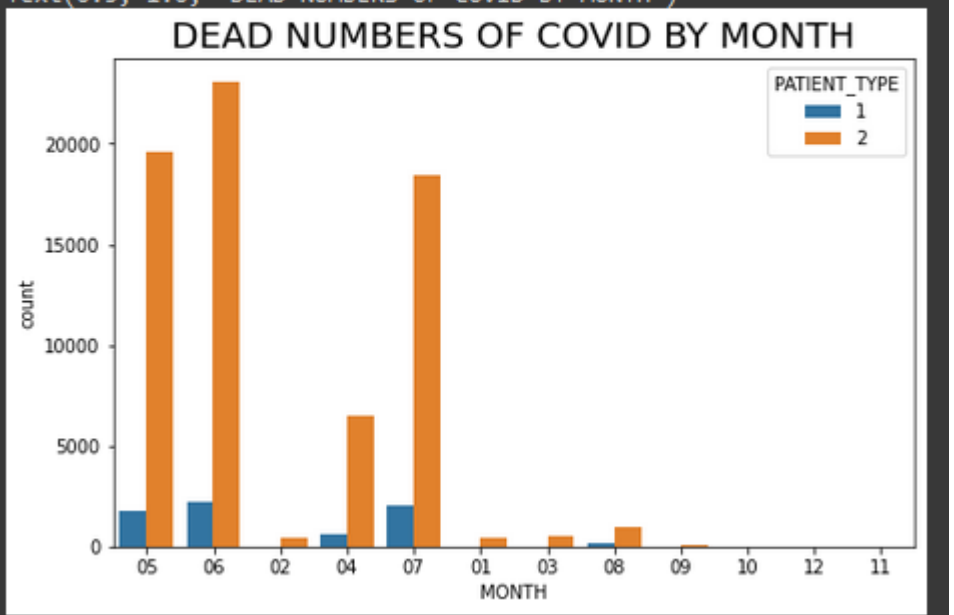


La siguiente gráfica de barras muestra las muertes por año.

- En 2020 alrededor de 68000 people murieron de COVID y no estaban hospitalizadas. Mientras que 6600 murieron de COVID y estaban hospitalizadas
- En 2021 alrededor de 60 personas murieron de COVID y no estaban hospitalizadas. Mientras que 250 personas murieron de COVID y estaban hospitalizadas.

```
plt.figure(figsize = (8,5))
ax = sns.countplot(x = "MONTH", hue = "PATIENT_TYPE", data = df)
plt.title("DEAD NUMBERS OF COVID BY MONTH", fontsize = 20)
```

```
Text(0.5, 1.0, 'DEAD NUMBERS OF COVID BY MONTH')
```



La siguiente gráfica muestra que los meses con más muertes fueron en Mayo, Junio y Julio encajando con el periodo vacacional. Por lo que es ideal que los gobernantes estén al pendientes de esas fechas en específico.

```
[10] df.drop(["DATE_DIED", "MONTH", "YEAR"], axis = 1, inplace = True)
```

Para comenzar con la clasificación se borrarán las columnas relacionadas con las fechas ya que nos interesa el análisis de los síntomas principalmente.

Antes de comenzar se hará un análisis de los valores que están en los registros.

```
[11] for i in df.columns:
         print(df[i].value_counts())
         print('\n')
```

```
6      40584
9      38116
3      19175
8      10399
10      7873
5       7244
11      5577
13       996
7        891
2        169
1        151
Name: MEDICAL_UNIT, dtype: int64


1    525064
2    523511
Name: SEX, dtype: int64


1    848544
2    200031
Name: PATIENT_TYPE, dtype: int64


2     892534
1     140038
99     16003
Name: PNEUMONIA, dtype: int64
```

```
30     27010
31     25927
28     25313
29     25134
34     24872
       ...
114        2
116        2
111        1
121        1
113        1
Name: AGE, Length: 121, dtype: int64


2    1040444
1       8131
Name: PREGNANT, dtype: int64


2     920248
1     124989
98      3338
Name: DIABETES, dtype: int64
```

```
2     1030510
1       15062
98       3003
Name: COPD, dtype: int64


2     1014024
1       31572
98       2979
Name: ASTHMA, dtype: int64


2     1031001
1       14170
98       3404
Name: INMSUPR, dtype: int64


2     882742
1     162729
98      3104
Name: HIPERTENSION, dtype: int64
```

```
2     1015490
1       28040
98       5045
Name: OTHER_DISEASE, dtype: int64


2     1024730
1       20769
98       3076
Name: CARDIOVASCULAR, dtype: int64


2     885727
1     159816
98      3032
Name: OBESITY, dtype: int64


2     1026665
1       18904
98       3006
Name: RENAL_CHRONIC, dtype: int64
```

```
2     960979
1      84376
98      3220
Name: TOBACCO, dtype: int64


2    656596
1    391979
Name: CLASIFFICATION_FINAL, dtype: int64


97    848544
2     175685
1      16858
99      7488
Name: ICU, dtype: int64
```

Existen algunas columnas con valores nulos, pero no son suficientes para estorbar en la clasificación. Sin embargo la columna de ICU tiene más registros nulos por lo que es mejor borrarla.

```
df.drop("ICU", axis = 1, inplace = True)
```

Además se le restará 1 a los valores actuales de la clasificación para que sea más entendible entre 1 y 0.

```
df["CLASIFFICATION_FINAL"] = df["CLASIFFICATION_FINAL"] - 1
```

Entrenamiento del Modelo

```
X = df.drop("CLASIFFICATION_FINAL", axis = 1)
y = df["CLASIFFICATION_FINAL"]
```

Se hace la división de las columnas entre la variable dependiente y la independiente.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

Se hace la división de prueba y entrenamiento.

```python
from sklearn.metrics import fbeta_score, accuracy_score
from time import time

def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    results = {}

    # Ajustar al alumno a los datos de entrenamiento usando cortes con 'sample_size'
    start = time() # Obtener hora de inicio
    learner = learner.fit(X_train[:sample_size],y_train[:sample_size])
    end = time() # Obtener hora de termino

    # Calcular el tiempo de entrenamiento
    results['train_time'] = end - start

    #  Get the predictions on the test set,
    #  then get predictions on the first 300 training samples
    start = time() # Obtener hora de inicio
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time() # Obtener hora de termino

   # Calcular el tiempo total de predicción
    results['pred_time'] = end - start

    # Calcular la precisión en las primeras 300 muestras de entrenamiento
    results['acc_train'] = accuracy_score(y_train[:300],predictions_train)

   # Calcular la precisión en el conjunto de prueba
    results['acc_test'] = accuracy_score(y_test,predictions_test)

    # Calcule la puntuación F en las primeras 300 muestras de entrenamiento
    results['f_train'] = fbeta_score(y_train[:300],predictions_train, beta = 0.5)

    # Calcule la puntuación F en el conjunto de prueba
    results['f_test'] = fbeta_score(y_test,predictions_test, beta= 0.5)

    # Éxito
    print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size))

    # Return the results
    return results
```

El siguiente método tiene como función hacer diferentes ejecuciones de modelos con el fin de obtener resultados de rendimiento.

```
# Importe los tres modelos de aprendizaje supervisado de sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier

# Inicialice los tres modelos, los estados aleatorios se establecen en 101
clf_A = DecisionTreeClassifier(random_state=101)
clf_B = BaggingClassifier(n_estimators=10, random_state=101)
clf_C = RandomForestClassifier(max_depth=2, random_state=101)
clf_D = GradientBoostingClassifier(random_state=101)
clf_E = AdaBoostClassifier(random_state = 101)
clf_F = LogisticRegression(random_state = 101)
clf_G = KNeighborsClassifier(n_neighbors = 3)
```

El siguiente código tiene todos los clasificadores a probar que incluyen algunos métodos de clasificación comunes y métodos de ensamblaje.

- Decision Tree
- Adaboost
- Bagging
- Random Forest
- Gradient Boosting
- Logistic Regression
- K-Neighbors

Se crean los diferentes clasificadores, con sus valores determinados.

```
# Calcule el número de muestras para el 1 %, 10 % y 100 % de los datos de entrenamiento
samples_1 = int(round(len(X_train) / 100))
samples_10 = int(round(len(X_train) / 10))
#samples_100 = int(round(len(X_train) / 2))
samples_100 = len(X_train)
# Recopilar resultados sobre los estimadores
results = {}
for clf in [clf_A, clf_B, clf_C, clf_D, clf_E, clf_F, clf_G]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = train_predict(clf, samples, X_train, y_train, X_test, y_test)
```

Luego se calculan 3 diferentes tipos de muestras, el 1, 10 y 100% de los datos del entrenamiento. Luego se hace un for loop para crear la matriz de resultados además de iterar por los diferentes clasificadores y ponerlos dentro de la misma.

Resultados

DecisionTreeClassifier

	1%	10%	100%
train_time	0.026892	0.245834	3.919235
pred_time	0.098926	0.123446	0.166283
acc_train	0.850000	0.746667	0.723333
acc_test	0.590089	0.628770	0.651162
f_train	0.862231	0.780834	0.756442
f_test	0.669888	0.692263	0.703687

BaggingClassifier

	1%	10%	100%
train_time	0.147699	1.805600	39.597378
pred_time	0.954979	1.388562	2.053326
acc_train	0.840000	0.753333	0.723333
acc_test	0.608024	0.633284	0.653273
f_train	0.849315	0.779777	0.753871
f_test	0.682210	0.694028	0.703826

El primer par a analizar es el Árbol de Decisión y Bagging. Los resultados son algo positivos en ambos. En general el árbol de decisión tiene mejores resultados pero no mucha diferencia. Sin embargo los tiempos de ejecución son menores en el árbol y se obtienen resultados similares.

RandomForestClassifier

	1%	10%	100%
train_time	0.255884	1.708136	23.163478
pred_time	1.826837	1.835397	1.901632
acc_train	0.650000	0.653333	0.653333
acc_test	0.657367	0.657453	0.657898
f_train	0.705378	0.707584	0.707584
f_test	0.699530	0.699680	0.700095

GradientBoostingClassifier

	1%	10%	100%
train_time	0.541554	5.637660	87.762473
pred_time	0.550271	0.598845	0.603579
acc_train	0.663333	0.656667	0.666667
acc_test	0.666208	0.668335	0.668811
f_train	0.716599	0.712551	0.718928
f_test	0.710032	0.711362	0.711612

En cuanto al Random Forest y Gradient Boosting los resultados son menores que los dos anteriores y con tiempos mayores de ejecución. Sin embargo el puntuaje F tiene resultados similares por lo que puede existir problemas en la precisión.

KNeighborsClassifier

	1%	10%	100%
train_time	0.003255	0.008304	0.058436
pred_time	48.076169	364.431275	3967.107620
acc_train	0.806667	0.790000	0.793333
acc_test	0.592006	0.595070	0.596459
f_train	0.838991	0.830861	0.830890
f_test	0.671052	0.673683	0.674784

LogisticRegression

	1%	10%	100%
train_time	0.182821	1.828080	17.174512
pred_time	0.041325	0.035250	0.038141
acc_train	0.643333	0.643333	0.643333
acc_test	0.661560	0.661748	0.661738
f_train	0.704453	0.704971	0.704971
f_test	0.706270	0.706387	0.706659

K-Neighbors tuvo resultados altos al momento de entrenar, sin embargo en los resultados de prueba fueron los más bajos. La regresión logística tiene resultados similares pudiendo indicar que los clasificadores tienen rendimientos similares. En base a estos resultados obtenidos se seguirá trabajando con el árbol de decisión. Por lo que se buscará utilizar los hiper parámetros.

Se especifican la mayor cantidad de parámetros disponibles para el árbol de decisión. Y se crea el clasificador nuevamente.

```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}
tree = DecisionTreeClassifier()
```

```
from sklearn.model_selection import GridSearchCV
tree_cv = GridSearchCV(tree, parameters, cv=10)
tree_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

Se ejecuta el GridSearch importando el método de la librería y se entrena. Se tienen en los resultados los parámetros que fueron ejecutados.

```
print("tuned hpyerparameters :(best parameters) ")
print(tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

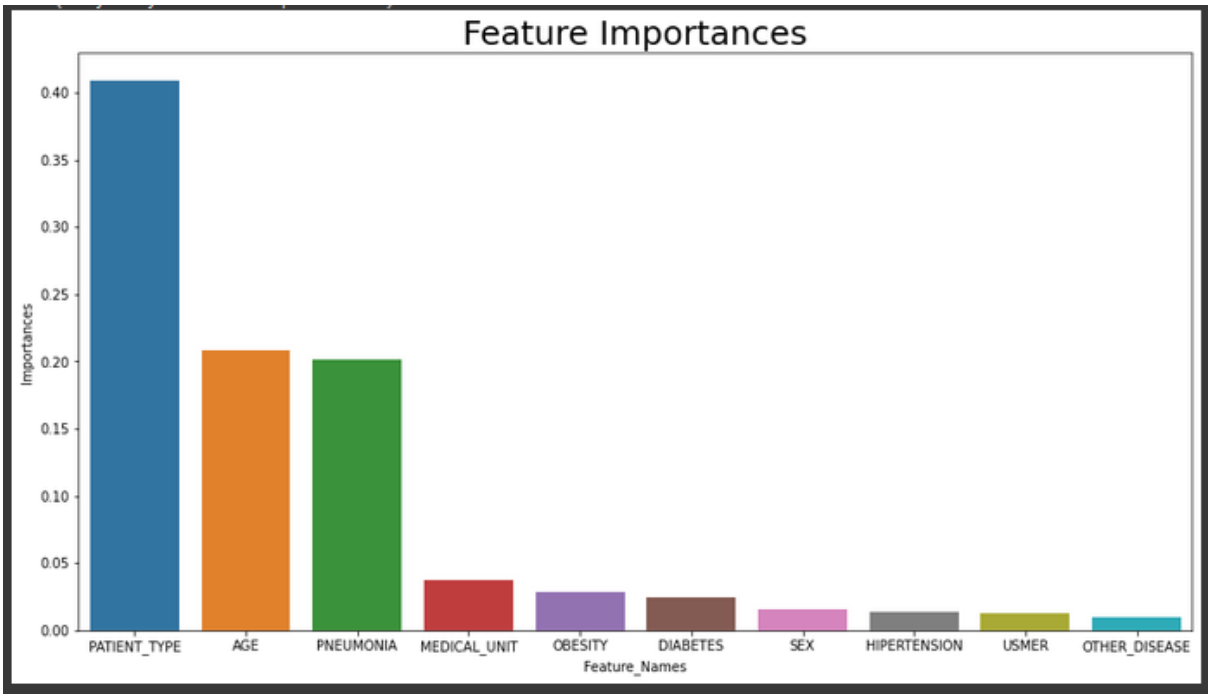
```
tuned hpyerparameters :(best parameters)
{'criterion': 'entropy', 'max_depth': 12, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter': 'best'
accuracy : 0.6664968259161805
```

Los parámetros ideales conseguidos fueron los siguientes, sin embargo el rendimiento no mejoró mucho en relación a los parámetros utilizados anteriormente. Teniendo en cuenta estos resultados se hará un método adicional para obtener los atributos que más influyen en las decisiones para clasificar.

```
[30] df_importances = pd.DataFrame(list(X), tree_cv.best_estimator_.feature_importances_)
     df_importances.columns = ["Feature_Names"]
     df_importances["Importances"] = df_importances.index
     df_importances = df_importances.sort_values(by = "Importances", ascending = False)
     df_importances.index = np.arange(0,len(df_importances))
```

```
[31] plt.figure(figsize = (15,8))
     ax = sns.barplot(x = "Feature_Names", y = "Importances", data = df_importances.sort_values(
         by = "Importances", ascending=False)[0:10])
     plt.title("Feature Importances", fontsize = 25)
```

Se obtienen los atributos más influyentes mediante feature_importances y se crean columnas para el nombre de la columna y la importancia que son números decimales que suman un total de uno y se grafica en una gráfica de barras.



Los resultados del feature importance parece por un énfasis en el tipo de paciente que se refiere si está hospitalizado o no y luego un resultado similar en cuanto a edad y neumonía que tienen una influencia similar. Debido a la naturaleza del árbol

de decisión es entendible que ponga un énfasis en estos tres parámetros ya que dependiendo si un paciente está hospitalizado o no influye o no de alguna manera que un paciente esté en condiciones óptimas mientras que la edad y la neumonía indican la posible vulnerabilidad ante los síntomas del COVID. Esto seguido de otros atributos como la obesidad y la diabetes. Algo interesante es que la unidad médica también influye de cierto modo indicando que hay sectores médicos del país que no están en condiciones para dar tratamiento adecuado.

Conclusiones

Como conclusión de los resultados se puede decir que el COVID tiene ciertas influencias no solo en cuanto a condiciones personales sino también en el ambiente laboral médico donde no existan condiciones para evitar propagaciones o tratar precondiciones que puedan llevar al contagio. Por lo que es necesario reforzar y mejorar las áreas de trabajo y seguir con las medidas de prevención principalmente para evitar contagios de enfermedades en general así reduciendo la cantidad de gente que requiera hospitalización. La conclusión de los modelos es que cada modelo tiene su manera de priorizar ciertos atributos por lo que la ejecución de otros modelos, a pesar de su nivel similar de rendimiento, puede dar como resultado otros atributos que influyen en las infecciones por lo que también se recomendaría el ejecutar otros modelos para observar la información que es obtenida. Como conclusión general se tiene que los modelos de clasificación tienen un papel fundamental en la recopilación y análisis de los datos debido a su capacidad de ejecutar y realizar algoritmos de forma rápida y precisa por lo que cada vez se busca mejorar las capacidades ejecutivas de cada uno. En general la automatización seguirá creciendo cada vez más eficiente para tomar las mejores decisiones a partir de los datos.

Referencias

- Borzyskowski, I., Mateen, B., Mazumder, A., & Wooldridge, M. (2021, June 1). *Data science and AI in the age of COVID-19*. The Alan Turing Institute. Retrieved from https://www.turing.ac.uk/sites/default/files/2021-06/data-science-and-ai-in-the-age-of-covid_full-report_2.pdf
- Mena, L. (2008, September 1). *Aprendizaje Automático a partir de Conjuntos de Datos No Balanceados y su Aplicación en el Diagnóstico y Pronóstico Médico*. Retrieved December 14, 2022, from https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/533/1/MenaCaLJ.pdf
- Fernandez, S. (2017, October 19). *SELECCIÓN DE VARIABLES PARA CLASIFICACIÓN NO SUPERVISADA UTILIZANDO UN ENFOQUE HÍBRIDO FILTER-WRAPPER*. Repositorio INAOE. Retrieved December 14, 2022, from https://inaoe.repositorioinstitucional.mx/jspui/bitstream/1009/604/1/SolorioFS.pdf
- *Información referente a Casos covid-19 en México - datos.gob.mx/busca*. de México - Información referente a casos COVID-19 en México. (n.d.). Retrieved December 14, 2022, from https://www.datos.gob.mx/busca/dataset/informacion-referente-a-casos-covid-19-en-mexico
- Gao, J., Fan, W., & Han, J. (2010, May 1). *On the Power of Ensemble: Supervised and Unsupervised Methods Reconciled*. CSE Buffalo. Retrieved December 14, 2022, from https://cse.buffalo.edu/~jing/sdm10ensemble.htm
- Baturalpsert. (2022, December 5). *Covid19 XGBOOST & Eda*. Kaggle. Retrieved December 14, 2022, from https://www.kaggle.com/code/baturalpsert/covid19-xgboost-eda/notebook