Tecnológico de Monterrey

Proyecto Integrador (Gpo. 10)

Avance 2. Ingeniería de características

Equipo #6

Julio César Pérez Zapata	A01793880
Christian Emilio Saldaña López	A00506509
Jorge Estivent Cruz Mahecha	A0179380

En esta fase, conocida como ingeniería de características (FE - *Feature Engineering)*:

A. Se aplicarán operaciones comunes para convertir los datos crudos del mundo real, en un conjunto de variables útiles para el aprendizaje automático. El procesamiento puede incluir:

Primer Modelo(SincNet)

- El primer modelo realizado, se hizo extrayendo las características mediante SincNet d-vectors:

  Se tienen los datos adquiridos del dataset Saarbruecken Voice Database los cuales se empezaron a manejar y analizar posteriormente al resultado del desbalance de las clases se tomó la decisión de hacer un proceso de normalización se procedió a hacer las siguientes pruebas:

  Un undersampling inicialmente igualando las clases la de menor cantidad de datos para poder ver normalizado el dataset y evaluar cómo se comportaba al clasificar, se utiliza una semilla random para agregar la aleatoriedad al tomar los datos, esto permite reducir sesgos humanos en el proceso de normalización.

```python
def cargar_datos_con_undersampling(ruta_datos_enfermos, ruta_datos_sanos):
    # Listar todos los archivos
    archivos_enfermos = [f for f in os.listdir(ruta_datos_enfermos) if f.endswith(".wav")]
    archivos_sanos = [f for f in os.listdir(ruta_datos_sanos) if f.endswith(".wav")]

    # Asegurarse de que la selección sea reproducible
    random.seed(42)

    # Realizar undersampling en la clase mayoritaria
    min_size = min(len(archivos_enfermos), len(archivos_sanos))
    archivos_enfermos = random.sample(archivos_enfermos, min_size)
    archivos_sanos = random.sample(archivos_sanos, min_size)

    # Cargar datos con undersampling
    datos = []
    etiquetas = []

    # Cargar datos de personas enfermas
    for archivo in archivos_enfermos:
        archivo_path = os.path.join(ruta_datos_enfermos, archivo)
        datos_audio = cargar_audio(archivo_path)
        datos.append(datos_audio)
        etiquetas.append(1)  # Etiqueta 1 para personas enfermas

    # Cargar datos de personas sanas
    for archivo in archivos_sanos:
        archivo_path = os.path.join(ruta_datos_sanos, archivo)
        datos_audio = cargar_audio(archivo_path)
        datos.append(datos_audio)
        etiquetas.append(0)  # Etiqueta 0 para personas sanas

    # Convertir listas a arrays de NumPy para su manejo en ML
    X = np.array(datos)
    y = np.array(etiquetas)
    # Ahora files_healthy y files_sick tienen el mismo número de archivos
    print(f"Total de archivos sanos seleccionados: {len(archivos_enfermos)}")
    print(f"Total de archivos enfermos seleccionados: {len( archivos_sanos )}")
    return X, y
```

Posteriormente se procede a probar un método de data augmentation mediante un proceso de oversampling agregando datos junto a la integración de datos con ruido blanco para poder igualar las clases y evaluar cómo se comportaba

```python
# Rutas de las carpetas con datos de audio
ruta_enfermos = "/content/drive/MyDrive/PROYECTO_FINAL/Frases_enfermos_HM/export"
ruta_sanos = "/content/drive/MyDrive/PROYECTO_FINAL/frases sanos + whitenoise sanos"

# Cargar datos
X, y = cargar_datos(ruta_enfermos, ruta_sanos)

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

La extracción de características se realizó directamente con una red pre entrenada(Usando Transfer Learning),cargando los datos crudos como

embeddings mediante el uso de la red SincNet d-vectors, esto nos permite el usar la red neuronal ya pre entrenada con una serie de filtros SincNet exporta correctamente estas características para poder realizar la correcta clasificación entre las etiquetas definidas(Enfermos y Sanos).

```
# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
```

```
# Definir la arquitectura del modelo
out_dim = 50  # número de clases

sinc_layer = SincConv1D(N_filt=64, Filt_dim=129, fs=31800, stride=16, padding="SAME")

inputs = Input((31800, 1))
x = sinc_layer(inputs)
x = LayerNorm()(x)

x = LeakyReLU(alpha=0.2)(x)
x = MaxPooling1D(pool_size=2)(x)

# Resto de las capas...

# Definición del modelo
x = Flatten()(x)
x = Dense(256)(x)
x = BatchNormalization(momentum=0.05, epsilon=1e-5)(x)
x = LeakyReLU(alpha=0.2)(x)

x = Dense(256)(x)
x = BatchNormalization(momentum=0.05, epsilon=1e-5)(x)
x = LeakyReLU(alpha=0.2)(x)
```

```
prediction = Dense(out_dim, activation='softmax')(x)
model = tf.keras.models.Model(inputs=inputs, outputs=prediction)

# Compilar el modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

Segundo Modelo

- Aquí el segundo modelo con CNN, extrayendo las características manualmente:

  Generación de nuevas características:

Inicialmente se estaba trabajando con la extracción de características de los MFCC, pero teniendo en cuenta lo solicitado en la ingeniería de características para buscar alter complejidad de los datos.

nativas de mejorar el modelo, se crearon nuevas características en función de los MFCCs extraídos,aunque los MFCC son características poderosas para representar información relevante en señales de audio, la ingeniería de características puede ayudar a mejorar aún más la capacidad del modelo para capturar patrones específicos o reducir la

Ahora presentamos resultados de precisión con las características iniciales(solo MFCC:

Vemos un resultados en la precisión del 64%

Ahora con las características adicionales incluidas:

El resultado fue casi el mismo (65%)sin una mejora significativa, esto quiere decir que las nuevas características generadas no son informativas o están altamente correlacionadas con otras características existentes, pueden introducir ruido y no aportar valor adicional al modelo.esto lo veremos en el siguiente punto.

Fotos de nuevas características.

```python
def extraer_caracteristicas_1(ruta_completa,clase):
    y, sr = librosa.load(ruta_completa)

    # Extracción de características
    mean = np.mean(y)
    std = np.std(y)

    #Calcular los Mel-Frequency Cepstral Coefficients (MFCC)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    #Obtener el promedio de los coeficientes MFCC
    mean_mfccs = np.mean(mfccs, axis=1)

    # Calcular las derivadas primera y segunda (delta y delta-delta)
    delta_mfcc = librosa.feature.delta(mfccs)
    delta_delta_mfcc = librosa.feature.delta(mfccs, order=2)

    #Características en el dominio del tiempo
    max_amplitude = np.max(np.abs(y))
    min_amplitude = np.min(np.abs(y))

    #Características en el dominio de la frecuencia
    spectral_centroid = np.mean(librosa.feature.spectral_centroid(y=y, sr=sr)[0])
    spectral_bandwidth = np.mean(librosa.feature.spectral_bandwidth(y=y, sr=sr)[0])
    spectral_rolloff = np.mean(librosa.feature.spectral_rolloff(y=y, sr=sr)[0])

    #Características de energía
    zero_crossing_rate = np.mean(librosa.feature.zero_crossing_rate(y=y)[0])
    rms = np.mean(librosa.feature.rms(y=y)[0])
    clase clase
```

```
[ ]
       Media señal     Dstandar       MFCC_1       MFCC_2       MFCC_3  \
count  1494.000000  1494.000000  1494.000000  1494.000000  1494.000000
mean     -0.003213     0.118521  -285.282135   108.009033    23.426401
std       0.019123     0.041163    38.660976    17.561779    11.090279
min      -0.157601     0.035656  -491.181824    47.859734   -12.093027
25%      -0.000067     0.088874  -311.854179    96.597408    16.050041
50%       0.000090     0.111920  -286.031967   107.195984    23.712968
75%       0.000310     0.140940  -260.272789   120.183561    31.122838
max       0.005689     0.379412  -155.457825   163.457306    58.902077

             MFCC_4       MFCC_5       MFCC_6       MFCC_7       MFCC_8  ...  \
count  1494.000000  1494.000000  1494.000000  1494.000000  1494.000000  ...
mean     29.827608    11.924997    -4.207934    -3.221340     4.052886  ...
std       8.629341     8.556303     9.034946     6.821115     6.858654  ...
min       1.086860   -21.603552   -30.078554   -26.732029   -19.999046  ...
25%      23.865721     5.831425   -10.469187    -7.999327    -0.566116  ...
50%      29.726661    11.849786    -5.012325    -2.870015     4.132020  ...
75%      35.921998    17.879104     1.736169     1.796740     8.475030  ...
max      57.282078    49.110023    24.169561    20.284966    27.437059  ...

            MFCC_11      MFCC_12      MFCC_13      Amáxima          Amín  \
count  1494.000000  1494.000000  1494.000000  1494.000000  1.494000e+03
mean     -7.799371    -1.734573    -7.277539     0.670993  7.298134e-07
std       5.088238     4.584546     4.827751     0.192341  2.833765e-06
min     -25.576405   -18.944057   -23.155867     0.168511  0.000000e+00
25%     -10.987303    -4.788635   -10.404715     0.522018  0.000000e+00
50%      -7.688388    -1.796845    -7.518412     0.656418  2.051820e-09
75%      -4.251037     1.400147    -4.047214     0.812010  5.456077e-07
max       8.904531    12.548744     6.672144     1.164480  6.192806e-05

              AvgCS       AvgBws   Avgrolloff    AvgCrossZ          RMS
count  1494.000000  1494.000000  1494.000000  1494.000000  1494.000000
mean   1673.351978  2124.531452  3311.323141     0.057829     0.099205
std     352.976694   294.459396   889.063608     0.017824     0.036036
min     667.745891  1398.945346  1277.545861     0.004014     0.022477
25%    1453.388717  1930.498906  2765.096156     0.047507     0.072976
50%    1645.774285  2093.870451  3155.426239     0.057452     0.092769
75%    1842.898772  2272.001783  3649.023377     0.066918     0.118473
max    3272.699674  3232.109353  7429.752604     0.170781     0.303140
```

Escalamiento (normalización, estandarización, min – max,…)

Se aplica una técnica de normalización mix-max scaling

El Min-Max Scaling es una técnica de normalización que transforma los datos a un rango específico, generalmente [0, 1]. Para cada característica, el valor mínimo de esa característica se asigna a 0, el valor máximo se asigna a 1, y los valores intermedios se escalan proporcionalmente.

```python
# Instanciar el MinMaxScaler
scaler = MinMaxScaler()

# Ajustar y transformar el DataFrame
df_normalizado = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

# Mostrar el DataFrame normalizado

print("\nDataFrame Normalizado:")
print(df_normalizado)
```

Dataframe sin estandarizar

```
              0          1          2          3          4          5   \
0    -276.717987   95.966431  16.797182  32.426895   5.460468 -14.558228
1    -331.671173   88.990852  26.850248  29.664236  18.703295  -9.027935
2    -298.162170   94.625618  47.661263  33.394974  14.282223   9.721114
3    -335.460968   90.559555  30.698204  34.027248  10.324809  -1.605529
4    -328.966431  125.024414  32.765697  21.456146   0.954221  -5.444850
...          ...        ...        ...        ...        ...        ...
3375 -299.999268   84.822945  30.384068  27.291128  15.361194   4.424277
3376 -254.731796   95.487541  38.923668  30.053751  14.873991   2.490798
3377 -259.503052   69.861420  29.047359  22.718458  -0.338743  -5.093669
3378 -241.017746   72.833458  22.816212  10.544170   1.574811  -7.388791
3379 -211.609833   82.958015  30.486120  35.612396  18.947086   3.773836

             6         7          8         9          10         11  \
0     0.648947   6.209880 -11.038387 -2.728484 -15.955387  -4.830564
1     0.670050   6.027178 -14.027277 -4.579039 -11.164975  -4.951526
2     3.715047  17.661226  -1.353177 -4.638551  -8.260089  -6.271485
3     6.281756  10.063773 -11.569776 -5.919106 -10.538960  -5.504457
4    -0.443363   2.875868  -8.184669 -8.114563  -7.164325 -11.196189
...        ...        ...        ...       ...        ...        ...
3375 -2.879648   7.122748  -4.835103 -7.628438  -7.981459   0.425982
3376 -5.420818   8.292114   2.471379 -0.328044  -7.534906  -3.788523
3377 -0.073438   0.348526  -2.738698 -5.018907  -6.781922   0.866892
3378  1.258463   3.783682 -14.781074 -9.954063  -7.996145  -0.830934
3379 -3.217421   8.508855  -4.661208 -3.813538  -9.394625  -4.030648

             12           13           14           15        16    17
0    -11.588970  1625.625807  1973.739746  2994.980315  0.062381  sick
1    -13.915579  1907.157887  2495.266379  4065.203272  0.062527  sick
2     -0.735719  1702.239366  2401.045299  3580.825966  0.046420  sick
3     -7.531395  1584.164885  1936.438780  2741.204364  0.062995  sick
4     -4.804162  1400.222067  1815.462351  2614.938354  0.056946  sick
...         ...          ...          ...          ...       ...   ...
3375  -5.635503  2166.070522  2622.576392  4588.391973  0.086749  sano
3376  -5.693202  2166.478623  2613.860657  4676.514799  0.078148  sano
3377  -8.918947  2313.376848  2659.623960  4911.749268  0.081682  sano
3378  -7.666831  2552.174848  2845.061153  5802.723245  0.093757  sano
3379  -5.809443  2220.044609  2581.448995  4706.253184  0.074509  sano
```

Dataframe estandarizado

```
DataFrame Normalizado:
            0         1         2         3         4         5         6   \
0     0.594933  0.494178  0.406932  0.557699  0.382727  0.286099  0.582363
1     0.442490  0.441898  0.548535  0.508538  0.570002  0.388043  0.582812
2     0.535446  0.484129  0.841668  0.574926  0.507481  0.733660  0.647576
3     0.431977  0.453655  0.602735  0.586178  0.451517  0.524867  0.702167
4     0.449994  0.711958  0.631857  0.362474  0.319002  0.454093  0.559131
...        ...       ...       ...       ...       ...       ...       ...
3375  0.530350  0.410661  0.598310  0.466308  0.522739  0.636019  0.507314
3376  0.655924  0.490589  0.718595  0.515469  0.515849  0.600378  0.453266
3377  0.642688  0.298530  0.579482  0.384937  0.300717  0.460567  0.566999
3378  0.693967  0.320804  0.491713  0.168294  0.327778  0.418259  0.595327
3379  0.775546  0.396684  0.599748  0.614386  0.573449  0.624029  0.500130

            7         8         9         10        11        12        13  \
0     0.552510  0.323840  0.539063  0.279024  0.448150  0.387786  0.272336
1     0.548659  0.255056  0.487618  0.417954  0.444309  0.309786  0.352379
2     0.793916  0.546728  0.485963  0.502200  0.402396  0.751647  0.294118
3     0.633754  0.311611  0.450364  0.436109  0.426752  0.523819  0.260548
4     0.482226  0.389513  0.389330  0.533979  0.246020  0.615251  0.208251
...        ...       ...       ...       ...       ...       ...       ...
3375  0.571754  0.466598  0.402844  0.510280  0.615062  0.587380  0.425991
3376  0.596406  0.634744  0.605795  0.523231  0.481238  0.585445  0.426107
3377  0.428947  0.514843  0.475389  0.545069  0.629063  0.477300  0.467871
3378  0.501363  0.237708  0.338192  0.509854  0.575151  0.519278  0.535764
3379  0.600975  0.470600  0.508898  0.469296  0.473550  0.581548  0.441336

            14        15        16   17
0     0.299133  0.253747  0.175117  1.0
1     0.570545  0.411869  0.175555  1.0
2     0.521510  0.340304  0.127229  1.0
3     0.279721  0.216252  0.176959  1.0
4     0.216763  0.197596  0.158811  1.0
...        ...       ...       ...  ...
3375  0.636799  0.489169  0.248229  0.0
3376  0.632263  0.502189  0.222422  0.0
3377  0.656079  0.536945  0.233028  0.0
3378  0.752584  0.668584  0.269255  0.0
3379  0.615396  0.506583  0.211505  0.0
```

Conclusión de normalización:

El resultado para ambos casos fue el mismo lo que quiere decir que la diferentes escalas no afectan al modelo, en poca palabra el modelo utilizado es capaz de relacionar las diferentes escalar

B. Además, se utilizarán métodos de filtrado para la selección de características y técnicas de extracción de características, permitiendo reducir los requerimientos de almacenamiento, la complejidad del modelo y el tiempo de entrenamiento. Los ejemplos siguientes son ilustrativos, pero no exhaustivos, de lo que se podría aplicar:

Además, se utilizarán métodos de filtrado para la selección de características y técnicas de extracción de características, permitiendo reducir los requerimientos de almacenamiento, la complejidad del modelo y el tiempo de entrenamiento. Los ejemplos siguientes son ilustrativos, pero no exhaustivos, de lo que se podría aplicar:

- Umbral de varianza
- Correlación
- Chi-cuadrado
- ANOVA
- Análisis de componentes principales (PCA)
- Análisis factorial (FA)

Primer Modelo(SincNet)

Con el fin de ayudar a reducir la complejidad del modelo y el tiempo de procesamiento se efectuó en si el transfer learning usando una red neuronal pre entrenada(SincNet) esto hace que se simplifique en gran medida el uso del modelo y es mucho más rápido para arrojar resultados específicos, dado que se aprovecha tanto la estructura como el tiempo invertido en un modelo previo que se diseñó en base a datasets con muchas más características que en sí el nuestro.

Solamente fue necesario ajustar la entrada al modelo(forma de los audios, Frecuencia y etiquetas), así como la caracterización de salida deseada(Selección Binaria Enfermos/Sanos) y el modelo con SincNet. Por lo mismo es el modelo que mejores resultados nos está brindando.

```
[ ]  accuracy = model.evaluate(X_test, y_test)[1]
     print(f'Accuracy en datos de prueba: {accuracy}')

     4/4 [==============================] - 0s 7ms/step - loss: 1.8493 - accuracy: 0.7323
     Accuracy en datos de prueba: 0.7322834730148315
```

* Todas las decisiones y técnicas empleadas deben ser justificadas.

Correlación segundo Modelo(CNN)

Para el filtrado de características se usó el mapa de calor de la matriz de correlación:

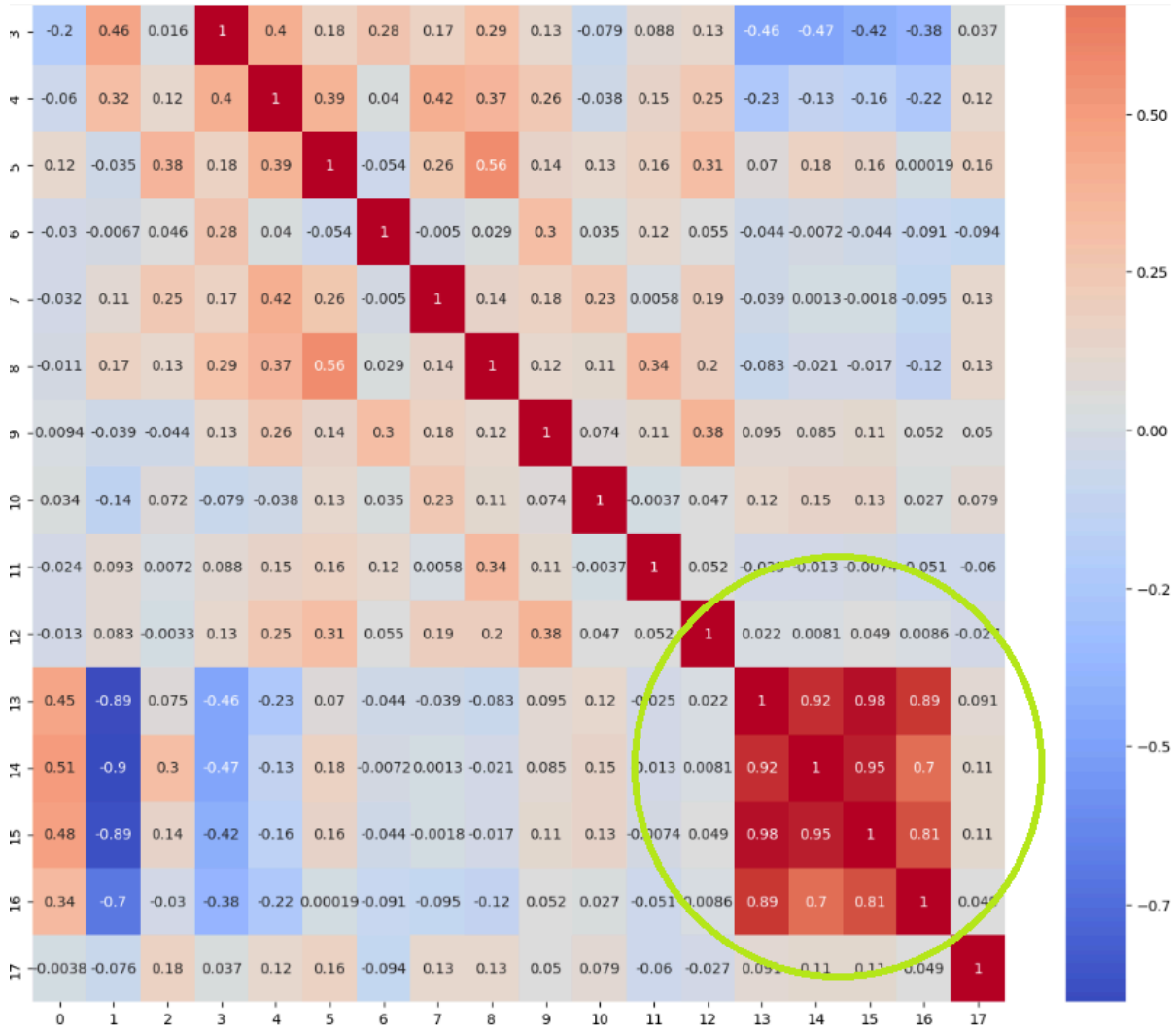
En esta vamos a ver que las característica adicionales a los MFCC están altamente correlacionadas entre sí, por lo que simplemente podemos dejar una sola variable, con esto reducimos la dimensionalidad del dataset y hacemos que el algoritmo sea más eficiente en términos de tiempo.

Para nuestro caso suprimimos las variables 13,15,16

En resumen, la matriz de correlación y su representación visual mediante un mapa de calor son esenciales para comprender las relaciones entre variables, identificar patrones, diagnosticar problemas en modelos y tomar decisiones informadas en el análisis de datos y la modelización estadística.

codigo:

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


      import pandas as pd
      import seaborn as sns
      import matplotlib.pyplot as plt



      corr_df = df.corr(method="pearson")

      plt.figure(figsize=(20, 15))
      sns.heatmap(corr_df, annot=True)
      # Crear una figura y ejes más grandes
      fig, ax = plt.subplots(figsize=(15, 15))

      # Crear el mapa de calor
      sns.heatmap(corr_df, cmap='coolwarm', annot=True, ax=ax)

      # Establecer el título
      plt.title('Mapa de Calor de Correlación', fontsize=16)

      # Mostrar el mapa de calor
      plt.show()
```



* Es necesario fundamentar los métodos ejecutados.

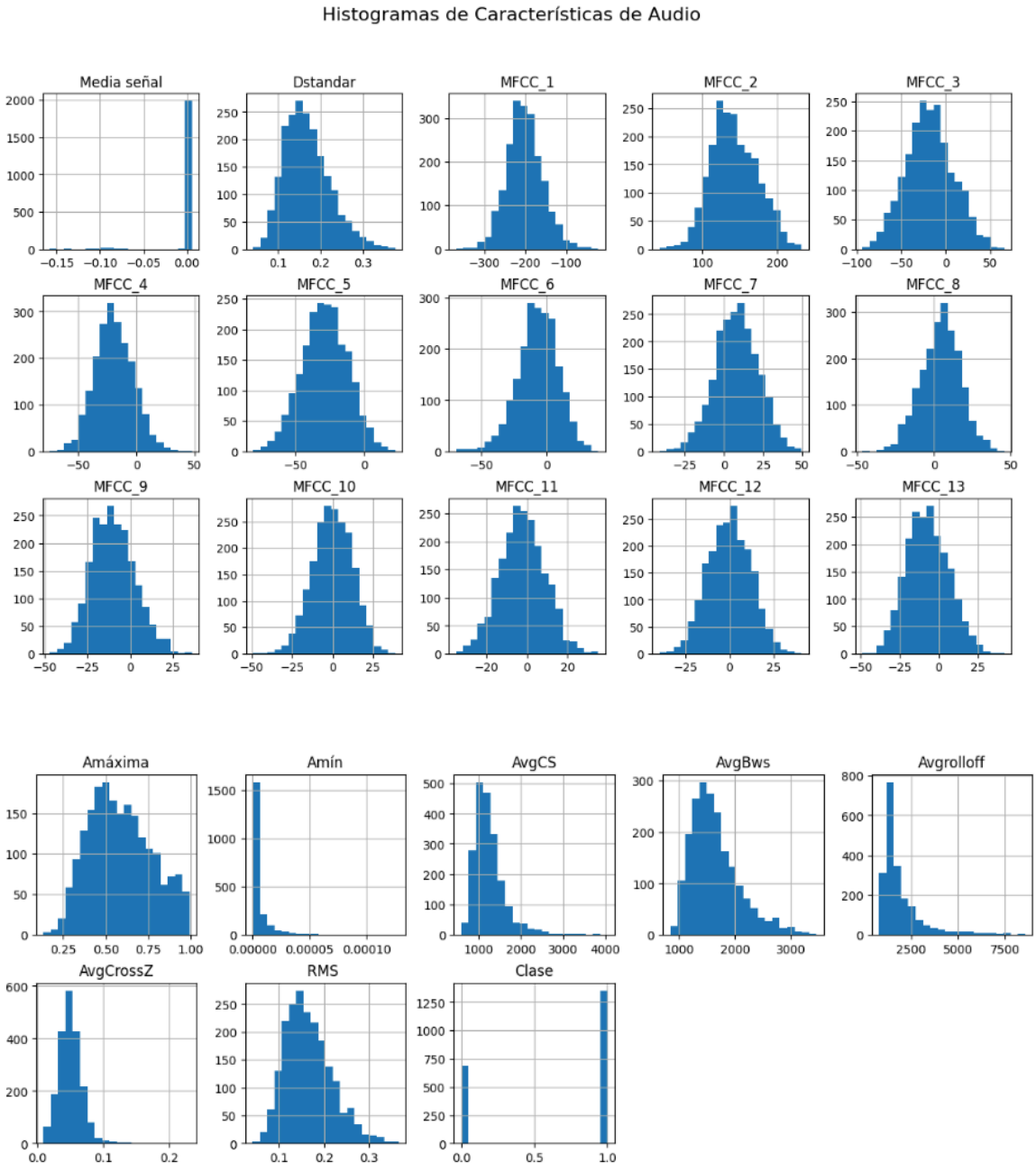
c. Incluir conclusiones de la fase de "Preparación de los datos" en el contexto de la metodología CRISP-ML.

Data Understanding and Exploration:

El dataset se encuentra compuesto de audios grabados en formato .wav los cuales fueron capturados por la Universidad de Saarlandes y separadas por diferentes etiquetas acerca de patologías vocales, el dataset está compuesto por grabaciones vocales de más de 2000 personas, las cuales contienen la grabación de las vocales a, i, u, adicionalmente de la frase “Guten Morgen, wie geht es ihnen”(Buenos dias, como estas.) grabadas en idioma alemán, para lo cual se realiza la transformación de los

datos(audios), extrayendo espectrogramas de mel para poder analizar las características de cada audio.

```
[51] # Visualizar histogramas
     df.hist(bins=20, figsize=(15, 15))
     plt.suptitle('Histogramas de Características de Audio', y=0.95, fontsize=16)
     plt.show()
```



Data Cleaning:

Durante el proceso se eliminaron varios audios que tienen muy poca duración (menos de 1 segundo). Debido a que no aportan soporte durante el entrenamiento.

Data Integration:

Se decidió agregar diferentes datasets para poder mejorar el entrenamiento. Originalmente utilizamos datasets de frases completas. Con la investigación se descubrió que utilizando audios de vocales también se podía calcular la patología del individuo.

Data Transformation:

Debido a que la clase de audios de gente sana resultó ser pequeña. Se hicieron transformaciones (data augmentation) con la técnica de white-noise para poder nivelar un poco los datos.

Data Splitting:

Se encontró que al tener 10% de datos para el test y el resto para entrenamiento se mejoró el ‘accuracy’.

Readiness for Modeling:

Actualmente se está considerando expandir el dataset con audios en EGG-signal. Para ver si se puede mejorar el ‘accuracy’ de los modelos. Como equipo creemos que los datos que se tienen actualmente si están listos para ser utilizados en modelos.

Github
https://github.com/julioperezzapata/Proyecto_integrador_grupo_6