cobertura o coverage, de una instancia es su vecindad; pero en vez de considerar un número fijo de vecinos, k, todas las instancias más cercanas que su enemigo más cercano está dentro de su conjunto de cobertura. El conjunto alcanzable o accesible (reachable) de una instancia es el conjunto de todas las instancias para las cuales, esa instancia particular, está dentro de su conjunto de cobertura.

La regla de eliminación es como sigue: Si la cardinalidad del conjunto accesible o alcanzable es mayor que su conjunto de cobertura la instancia es eliminada del dataset.

Algorithm 4: Iterative Case Filtering (ICF)

Input: A training set $X = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\}$, the number of nearest neighbours k
Output: The set of selected instances $S \subseteq X$

```
1  S = X
2  Noise filtering: remove any instance in S misclassified by its k neighbours
3  repeat
4      foreach instance x ∈ S do
5          Compute coverage(x)
6          Compute reachable(x)
7      progress = False
8      foreach instance x ∈ S do
9          if |reachable(x)| > |coverage(x)| then
10             Flag x for removal
11             progress = True
12     foreach instance x ∈ S do
13         if x flagged for removal then
14             S = S − {x}
15 until not progress
16 return S
```

Figura 25. Iterative Case Filtering. (ICF). Fuente: elaboración propia.

Discretización y compartimiento (binning)

La discretización es una de las técnicas de reducción de datos básicas. El proceso de discretización transforma datos cuantitativos en cualitativos, es decir, convierte atributos numéricos en atributos discretos o nominales con un número finito de intervalos, obteniendo una partición del dominio continuo. Luego se establece una asociación entre en cada intervalo y un valor discreto.

En la práctica, la discretización puede verse como un método de reducción de datos, debido a que mapea los datos correspondientes a un gran espectro de valores numéricos a un subconjunto reducido de valores discretos.

La discretización podría definirse de la siguiente forma: Supongamos que tenemos un dataset con N instancias y A atributos numéricos, entonces el algoritmo de discretización convertiría los atributos numéricos en un conjunto de m intervalos discretos $D = (d_0, d_1), (d_1, d_2), ..., (d_{m-1}, d_m)$ Donde d_0 es el valor mínimo del atributo y d_m el máximo. Mientras que $d_i < d_{i+i}$ para todo i=0,1...m-1. A D se le llama esquema de discretización del atributo A y $P = (d_1, d_2, ..., d_{m-1})$ el conjunto de puntos de corte.

Las motivaciones para discretizar pueden ser variadas, entre ellas que muchos algoritmos de aprendizaje automatizado trabajan solamente con rasgos o atributos nominales u otros solo con discretos. También, se sabe que hay algoritmos de aprendizaje cuyo rendimiento se ve afectado por el uso de variables que tienen distribuciones de probabilidad no estándar. Uno de los enfoques de solución sería convertir dichas variables a ordinales.

A este proceso de discretización se le conoce también como binning (Compartimentación, por darle algún nombre en español) y de lo que trata es de trasladar los valores de una variable cuantitativa a dos o más compartimentos o categorías. Los valores de la variable se agrupan en compartimentos o intervalos discretos, llamados bin y a cada uno se le asigna un único valor entero tal que se pueda mantener la relación de orden entre ellos.

Las estrategias que se pueden usar para acometerlo se pueden clasificar en:

- Uniforme: Cada bin tiene el mismo ancho.
- Quantile: Cada bin tiene el mismo número de valores.

- Clustered: Se identifican los grupos o clusters y los ejemplos se asignan a ellos.

Como en casi todos los ejemplos que presentamos a lo largo de esta lección, el módulo Scikit-Learn de Python viene a salvarnos.

Este módulo tiene una clase llamada KbinsDiscretizer, que permite implementar el proceso de discretización. Veremos un ejemplo; pero no profundizaremos en su uso.

```
from numpy.random import randn
from sklearn.preprocessing import KBinsDiscretizer
from matplotlib import pyplot
# genera una muestra de datos con distribución gaussian
data = randn(1000)
# Crea un histograma de los datos generados usa 25 bin
pyplot.hist(data, bins=25)
pyplot.show()
data = data.reshape((len(data),1))
# Crea el discretizador
kbins = KBinsDiscretizer(n_bins=10, encode='ordinal', strategy='uniform')
# Discretiza los datos usando 10 bin y una estrategia uniforme, retorna un ordinal
data_trans = kbins.fit_transform(data)
# Crea un histograma de los datos discretizados usa 10 bin
pyplot.hist(data_trans, bins=10)
pyplot.show()
```

Figura 26. Ejemplo de uso de un discretizador del módulo Scikit-learn. (KbinsDiscretizer). Fuente: elaboración propia.

Los histogramas generados por este ejemplo se muestran en las siguientes imágenes.

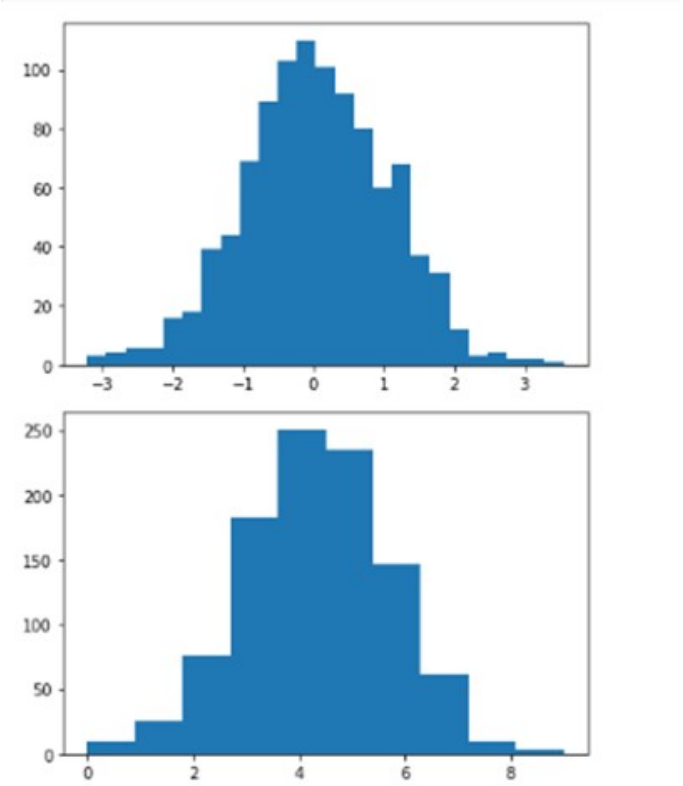

Figura 27. Resultados generados por el ejemplo de uso de un discretizador del módulo Scikit-learn. (KbinsDiscretizer). Fuente: elaboración propia.

Para precisar las opciones que brinda este discretizador, es necesario consultar la documentación del módulo Scikit-learn.

Hasta aquí hemos visto lo esencial de esta técnica y con eso concluimos esta lección.

Conclusiones

Hasta aquí hemos tocado aquellos conceptos y técnicas que componen una parte de la etapa de preprocesamiento de los datos y que antecede a la de entrenamiento de los modelos aprendizaje que escojamos para entrenar.

Para la próxima lección hemos dejado aquellos conceptos como balanceo de datos, separación en conjunto de entrenamiento y de prueba y lo relacionado con el análisis

exploratorio de datos que de una forma u otra va descubriendo los problemas que se dan en los conjuntos de datos y para resolverlos entonces usamos las técnicas o métodos que hemos visto en mayor o menor profundidad en esta lección.

Toda la documentación usada está disponible en Internet y los ejemplos de código desarrollados en Google Colab y Kaggle. El campo de aprendizaje automático es basto y complejo, no importa que de momento no entiendan todos los conceptos, en la medida que busque información sobre ellos y lea va ir viendo como encajan unos con otros.

Todo conocimiento que necesita se construye como una casa, piedra sobre piedra, concepto sobre concepto.

Tema 2. Análisis exploratorio de datos y preprocesamiento

2.3. Análisis exploratorio de datos y preprocesamiento II

Objetivos

El objetivo de esta lección es continuar con todo lo referente a la etapa de preprocesamiento de los datos y los conceptos conexos y que forman la base necesaria para la comprensión del proceso de aprendizaje automático. En esta parte incluimos conceptos como «separabilidad de las clases de problemas de clasificación» o la división del conjunto de datos en conjunto de entrenamiento o conjunto de prueba, que son cruciales a la hora de entender el resto de los conceptos. Incluimos también, en esta lección, los conceptos de Análisis Exploratorio de Datos y aquellos que proceden de la estadística descriptiva como medidas de tendencia central y varianza y covarianza. También incluimos las pruebas de normalidad para las variables o rasgos.

Al finalizar esta lección, estará equipado con los conceptos que son necesarios para entender y usar de manera práctica el aprendizaje automático.

Introducción

En la lección anterior vimos los problemas que se presentan en la recolección e integración de los datos y el tratamiento de los problemas que nos podemos encontrar. Vimos, por ejemplo, la detección de valores faltantes y los métodos de lidiar con ellos, así como la detección de datos extremos o anómalos que se le llaman Outliers. Tratamos, también en la lección anterior, técnicas para la selección de subconjuntos óptimos o no, con el fin de aliviar la carga que significa el anejo de cientos de rasgos, llamadas técnicas de selección de rasgos. Revisamos, las técnicas de selección de casos y que disminuyen o condensan un conjunto de datos.

Sin embargo, hay un problema que aún no se ha tocado y que puede afectar la calidad de la clasificación en modelos y es el correspondiente al desbalance entre la cantidad de casos pertenecientes a cada clase en un conjunto de datos.

Técnicas de tratamiento de conjuntos desbalanceados

Es bien conocido que los algoritmos de aprendizaje automáticos favorecen de alguna manera a las clases más pobladas, afectando la calidad de la clasificación y por tanto la habilidad de generalización de estos. Este problema de desbalance entre el número de casos en un dataset es muy común en la vida real y aparece, por ejemplo, en los conjuntos de datos en el área de salud, donde se pueden recolectar muchos casos de personas sanas y menos de enfermos. Pueden aparecer, también, en detección de fraudes, pues normalmente se recolectan muchas muestras de transacciones no fraudulentas y mucho menos de fraudulentas.

Con el fin de solucionar el efecto de este tipo de problemas en conjuntos de datos no balanceados se han desarrollado, a grandes rasgos dos grupos de técnicas:

- Técnicas a nivel de datos.
- Técnicas a nivel de algoritmos.

Las técnicas a nivel de datos pueden dividirse en:

- Muestreo de datos *(data sampling).*
- Selección de rasgos (no veremos ninguna aquí).

Las técnicas de muestreo de datos *(data sampling),* se dividen en dos grupos:

- Sobremuestreo (*Oversampling* e incluye SMOTE).
- Bajomuestreo *(undersampling).*

Por otra parte, las técnicas a nivel de algoritmos se pueden dividir en:

- Costo-sensibles *(cost-sensitive).*
- Híbridas/ensamble.

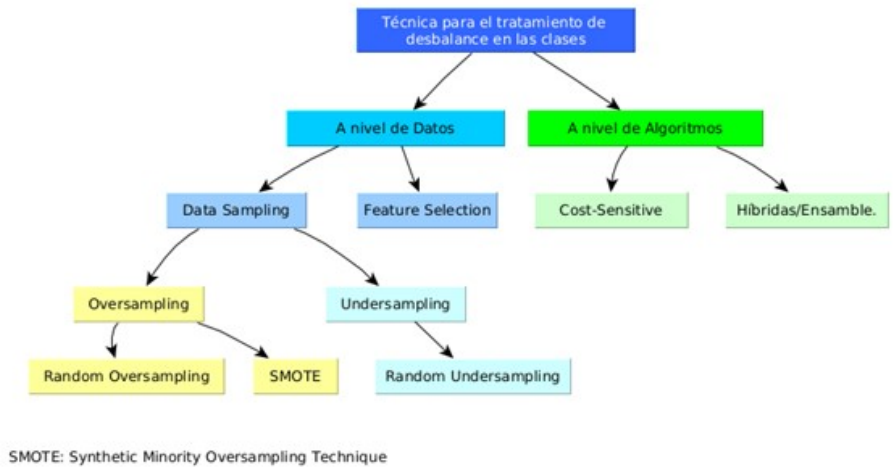

Figura 1. Clasificación de las técnicas de tratamiento de desbalance en las clases. Fuente: elaboración propia.

Las técnicas más usadas y con mejores rendimientos son las Data Sampling y por tanto son las que tocaremos aquí.

Básicamente, las técnicas que se engloban bajo este nombre de Data sampling, se basan en añadir o eliminar instancias de las clases minoritarias o mayoritarias ya sea de forma aleatoria o por algún método inteligente. Se le llama clase mayoritaria a aquella que tiene un número mayor de casos y minoritaria a las que están menos pobladas.

Random Undersampling: la idea detrás de este tipo de método es seleccionar ejemplos, de manera aleatoria, de la clase mayoritaria y eliminarlos del conjunto de entrenamiento. Este proceso puede repetirse de manera continuada hasta que las clases alcancen el balance. Es decir, las clases contengan el mismo número de

casos.

El problema con este tipo de método es que, al eliminar casos, podemos perder información valiosa o aún peor, crítica para la construcción de las fronteras de decisión entre clases.

Random Oversampling: la idea detrás de este método, es añadir o duplicar de manera aleatoria los casos de la clase minoritaria en el conjunto de entrenamiento.

Cuando se usa este método se incurre en el peligro de provocar un sobreajuste (overfitting) en nuestro modelo ya que este tipo de técnica hace copias exactas de los ejemplos de la clase minoritaria. Así que hay que tomar precaución a la hora de utilizarlo.

Synthetic Minority Oversampling Technique(SMOTE): este método es el estándar por defecto de este tipo de técnica. A diferencia del método original de Oversampling, que duplica los casos de la clase minoritaria, este sintetiza ejemplos y los añade al conjunto de entrenamiento por medio de interpolación. Eliminando así el peligro de que aparezca overfitting.

Con vistas a ejemplificar el uso de cada una de estas técnicas, apelaremos a un módulo de Python que implementa la mayoría de estas.

El dataset que usaremos para explicar cada una de estas técnicas es el Pima indian diabetic dataset. Este dataset tiene 768 casos y 9 rasgos. De los 768 casos, 268 son diabéticos y los no diabéticos 500. Lo que hace a este un dataset con desbalance moderado en las clases.

La librería que usaremos para los ejemplos se llama Imbalanced-Learn y se puede instalar de la siguiente manera:

```
sudo pip install imbalanced-learn
```

Esta librería contiene un grupo de clases que permiten implementar estas

estrategias, veamos.

define la estrategia de oversampling

```
oversample = RandomOverSampler(sampling_strategy='minority')
```

Este tipo de estrategia hace que si la clase mayoritaria tiene 1000 casos y la minoritaria 100, añade casos duplicados a esta hasta que alcance, también los 1000 casos. En el argumento sampling_strategy, en vez de una cadena se puede pasar un número de punto flotante que especifica la cantidad de casos de la clase minoritaria respecto a la mayoritaria. Por ejemplo, si se le pasa 0.5, indica que la clase minoritaria tendrá la mitad de los casos que la mayoritaria.

```
import pandas as pd
import numpy as np
from imblearn.over_sampling  import RandomOverSampler
#*******************************************************
diabeteDF = pd.read_csv("diabetes.csv")
print("Diabéticos:"+str(len(diabeteDF[diabeteDF.Outcome==1])))
print("No Diabéticos:"+str(len(diabeteDF[diabeteDF.Outcome==0])))
#*******************************************************
Y = diabeteDF["Outcome"]
X= diabeteDF.drop("Outcome",axis=1)
#*************************************************************
inbalaceProcesor = RandomOverSampler(sampling_strategy='minority')
X_Processed, Y_Processed=inbalaceProcesor.fit_resample(X,Y)
X_DF = pd.DataFrame(X_Processed, columns=['Pregnancies','Glucose','BloodPressure',
                                          'SkinThickness','Insulin','BMI',
                                          'DiabetesPedigreeFunction','Age'])
Y_DF = pd.DataFrame(Y_Processed, columns=['Outcome'])
diabete_Res = pd.concat([X_DF,Y_DF],axis=1,)
print("Diabéticos con resampling:"+str(diabete_Res[diabete_Res.Outcome==1].shape[0]))
print("No Diabéticos con resampling:"+str(diabete_Res[diabete_Res.Outcome==0].shape[0]))
```

```
Diabéticos:268
No Diabéticos:500
Diabéticos con resampling:500
No Diabéticos con resampling:500
```

Figura 2. Ejemplo de uso de la técnica de Random Oversampling, con el módulo imbalanced-learn. La estrategia usada es minority. Fuente: elaboración propia.

En la figura correspondiente al ejemplo de uso de la clase RandomOverSampler, hemos establecido la estrategia de muestreo en minority; pero esta acepta las siguientes estrategias:

- 'minority': Hace un re-muestreo solo de la clase minoritaria

- 'not minority': Hace un re-muestreo de todas las clases excepto la minoritaria.
- 'not majority': Hace un re-muestreo de todas las clases excepto la mayoritaria.
- 'all': Hace un re-muestreo de todas las clases;
- 'auto': Es equivalente a 'not majority'.
- No vamos a mostrar otro ejemplo para demostrar el uso de RandomUndersampler o SMOTE; porque el código es prácticamente el mismo, solo cambian las clases a utilizar y los resultados el mismo. Queda a los lectores explorar y experimentar con los parámetros de cada una de estas técnicas.
- Es importante que se revise la documentación del módulo imbalanced-learn, pues contiene algunas variantes a SMOTE, tales cómo SMOTENC, SMOTEN, ADASYN, BorderlineSMOTE, KmeansSMOTE y SVMSMOTE.

División del conjunto de datos. (Entrenamiento y prueba)

Hasta este momento hemos estado hablando de conjunto de entrenamiento o conjunto de prueba; sin embargo, no hemos dedicado ningún espacio a explicar los motivos por los que es necesaria esta separación.

En esta sección iremos explicando los motivos y haremos un ejemplo para que se entienda de manera práctica cómo se hace.

La intuición detrás de esta separación es probar cuan bien nuestros modelos generalizan. Veamos, si tomamos en cuenta que la mayoría de los modelos se construyen a partir de los conjuntos de datos que les damos entonces es buena idea dejar una parte de datos que no se usen para construir el modelo y usarlos para probar cuan bien generalizan.

De esta forma, podemos dividir el conjunto de datos que tenemos usando reglas como 80-20 para la división; pero esto no es fijo. También podría usarse 75-25, u otra. Esto quiere decir que usaríamos el 80% de los datos para construir el modelo y

20 para probar su habilidad de generalización.

La idea detrás del entrenamiento es disminuir la cantidad de errores en el conjunto de entrenamiento, con la esperanza que esto nos lleve al mínimo de errores en el conjunto de prueba. Pero en la realidad es mucho más complejo y los detalles los veremos en la parte correspondiente a la evaluación de los modelos de aprendizaje automático.

En la literatura es común encontrarse con divisiones del tipo: Entrenamiento, Validación y Prueba. Lo cierto es que dividimos en entrenamiento y prueba, porque los algoritmos de entrenamiento usan el conjunto, y valga la redundancia, de entrenamiento para dividirlo por medio de una técnica que se llama validación cruzada en dos más: Entrenamiento y validación.

En esencia, solo dividimos en dos conjuntos: Entrenamiento y Prueba. Entonces el algoritmo de entrenamiento, al que le pasamos el primero, a su vez los divide según la validación cruzada en: Entrenamiento y validación; pero creo eso lo había dicho y solo lo repito con el propósito de reforzar la idea.

La manera más sencilla de hacer esta división sería hacerlo de forma lineal, es decir tomar el conjunto de datos y dividirlo a partir de una fila determinada, sin embargo, esta no es buena idea.

Lo ideal sería, si tenemos dos clases, por ejemplo y la emparejamos por medio de re-muestreo, tomar de forma aleatoria la misma proporción de ambas en forma de 80-20. Esto nos garantiza que no introduzcamos ningún sesgo en el conjunto escogido y las clases tendrás la misma representación en ambos conjuntos. A este tipo de procedimiento se le conoce como muestreo aleatorio estratificado.

Esto es exactamente lo que hace la clase train_test_split del módulo Scikit-Learn de python y que usaremos para ejemplificar el procedimiento.

Es importante dejar claro que el balanceo de las clases por re-muestreo debe ejecutarse sobre el conjunto de entrenamiento.

Si queremos que train_test_split, deje de hacer muestreo aleatorio estratificado le pasamos como parámetro shuffle = False.

El tamaño del conjunto de entrenamiento o prueba se le pasa por medio del parámetro train_size o por medio de test_size, se puede definir uno solo y automáticamente se asume que el otro es el complemento.

Es decir, si se le pasa como argumento test_size=.25 eso hace que el conjunto de entrenamiento sea del 75% de los casos del dataset.

```
import pandas as pd
import numpy as np
from imblearn.over_sampling  import SMOTE
from sklearn.model_selection import train_test_split
#*********************************************************
diabeteDF = pd.read_csv("diabetes.csv")
print("Diabéticos:"+str(len(diabeteDF[diabeteDF.Outcome==1])))
print("No Diabéticos:"+str(len(diabeteDF[diabeteDF.Outcome==0])))
#*********************************************************
Y = diabeteDF["Outcome"]
X = diabeteDF.drop("Outcome",axis=1)
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=.25, random_state=0)
#*************************************************************
# Debe tenerse como norma que el balanceo de las clases debe ejecutarse sobre el conjunto de entrenamiento
inbalaceProcessor = SMOTE()
X_Processed, Y_Processed=inbalaceProcessor.fit_resample(X_train,Y_train)
X_DF = pd.DataFrame(X_Processed, columns=['Pregnancies','Glucose','BloodPressure',
                                          'SkinThickness','Insulin','BMI',
                                          'DiabetesPedigreeFunction','Age'])
Y_DF = pd.DataFrame(Y_Processed, columns=['Outcome'])
diabete_Res = pd.concat([X_DF,Y_DF],axis=1,)
print("Diabéticos con resampling usando SMOTE:"+str(diabete_Res[diabete_Res.Outcome==1].shape[0]))
print("No Diabéticos con resampling usando SMOTE:"+str(diabete_Res[diabete_Res.Outcome==0].shape[0]))
```

```
Diabéticos:268
No Diabéticos:500
Diabéticos con resampling usando SMOTE:370
No Diabéticos con resampling usando SMOTE:370
```

Figura 3. Ejemplo de división del conjunto de datos en: Conjunto de entrenamiento(X_train,Y_train) y Conjunto de Pruebas (X_test, Y_test). Fuente: elaboración propia.

En la sección correspondiente a las métricas de evaluación de los modelos de aprendizaje, también detallaremos todo lo que respecta a la validación cruzada y como ejecutarla con Scikit-Learn.

Análisis exploratorio de datos

Desde la sección anterior y la parte inicial de esta, hemos estado analizando técnicas

y métodos que corresponden a la etapa de preprocesamiento de los datos en el proceso de aprendizaje automático. Sin embargo, es durante el proceso de análisis exploratorio donde se detectan la mayoría de los problemas en nuestro conjunto de datos y que se resuelven o atacan con las técnicas que ya hemos visto. Esto quiere decir que Análisis Exploratorio de Datos está íntimamente ligada a la etapa de preprocesamiento de los datos, toda vez que nos permite ir visualizando, detectando y corrigiendo los problemas que pueden presentarse en dicha etapa.

¿Qué es el Análisis Exploratorio de Datos? Es un conjunto de técnicas que nos permiten entender los datos que tenemos a mano, detectar anomalía en ellos y corregirlas. De manera general, esta etapa pretende adaptar los conjuntos de datos a las necesidades de los modelos de aprendizaje que usaremos para resolver las tareas a mano.

Cada conjunto de datos es único y por tanto es muy difícil automatizar todas las tareas referente a su chequeo, análisis y corrección, por eso es importante un conjunto de pasos ordenados y técnicas definidas para su tratamiento, eso es el Análisis Exploratorio de datos.

Etapas de análisis exploratorio de datos

Las etapas que, de manera general, se llevan a cabo en AED, son las siguientes:

- Preparación de los Datos.
- Realizar una inspección general del conjunto de datos y de sus rasgos.
- Realizar un análisis de las correlaciones entre las variables o rasgos que conforman el conjunto.
- Detección de los valores faltantes y su corrección-
- Detección y corrección de datos anómalos, los llamados Outliers.

- Realizar una evaluación, si fuese necesario, de los supuestos básicos de las técnicas estadísticas que deben cumplir las variables tales como Normalidad, linealidad y homocedasticidad.

Lo importante a destacar aquí, es que esta no es una etapa que debe seguirse de forma lineal, puede verse como una etapa iterativa.

Aunque este AED, puede hacerse con herramientas gráficas tales como SPSS, STATA u otros, a los efectos de este trabajo usaremos Python y sus módulos para acometerlo.

Cómo ya hemos visto muchas de las técnicas que se deben aplicar en esta etapa, pues nos concentraremos en la visualización y tipos de gráficas que son útiles para el análisis, los conceptos relacionados con las distribuciones de frecuencias básicas, las pruebas de normalidad y homocedasticidad.

Teniendo en cuenta que las técnicas básicas para la integración de datos ya las vimos en secciones anteriores no nos detendremos a explicarlas y supondremos que tenemos un archivo ya integrado en formato CSV, es decir, de valores separados por coma.

Ahora bien, es de vital importancia estar familiarizado con los conceptos básicos de la estadística descriptiva y los gráficos más comunes para mostrar información y sus usos. Por tanto, antes de ver los detalles de cada una de las etapas del proceso, veremos los conceptos básicos de AED.

Conceptos de la estadística descriptiva básicos

Los conceptos que trataremos en este apartado son los usados para describir a una variable en estadística.

Medidas de tendencia central

Dentro de las medidas que permiten sumariar un rasgo o variable, se encuentran:

- Media.
- Mediana.
- Moda.

Comencemos por la más simple, la media:

La media es un número que caracteriza el centro de una colección de valores, por ejemplo, un rasgo. Es a lo que comúnmente se conoce como promedio.

Si tomamos como ejemplo los siguientes valores: 3,5,1,2. Entonces la media sería la suma de ellos dividida por la cantidad.

(3+5+1+2) / 4=11/4 = 2.75

Su fórmula matemática es:

$$\bar{x} = \frac{1}{n}\sum_{1}^{n} x_i$$

Lo importante a notar de la media es que es muy susceptible a valores anómalos u Outliers, y puede ser engañosa cuando no se han detectado y tratados de forma correcta.

El cálculo de la mediana en Python para un rasgo como la glucosa, del dataset de diabetes sería:

Por ejemplo: print("Media:"+str(np.mean(diabete_Res["Glucose"])))

Aquí se usa la librería Numpy.

Mediana

Esta es otra de las medidas que caracterizan el centro de los valores de una variable. Esta se define como el valor del centro o medio de una variable ordenada. Si la

variable es impar, la mediana coincide con el valor central. Si la variable es par entonces la mediana sería el promedio de los dos valores del centro.

Si, tomamos como ejemplo el mismo conjunto de valores anteriores: 3,5,1,2.

La mediana sería: 1,2,3,5 = 2+3/2 = 2.5.

Si el conjunto estuviera compuesto por un número impar de valores:

mediana = 1,2,3,5,6 = 3

La mediana es una medida de centralidad mucho más robusta a outliers que la media.

Introduzcamos a manera de ejemplo un outlier en el conjunto anterior y veamos cómo se comporta la mediana y la media.

mediana = 1,2,3,5,12 = 3

media = (3+5+1+2+12) /5=23/5 = 4.6

El cálculo de la mediana en Python para un rasgo como la glucosa, del dataset de diabetes sería:

```
print("Media:"+str(np.median(diabete_Res["Glucose"])))
```

Aquí se usa la librería Numpy.

Moda

La Moda es el valor que se repite con mayor frecuencia en un conjunto de datos. Cuando un conjunto de datos se representa por medio de las frecuencias de los valores que contiene, a esa representación se le llama distribución de frecuencias. Cuando en la distribución de frecuencias, aparece un solo valor con la máxima frecuencia, se le llama unimodal. Pues contiene una moda. Pueden aparecer varios valores con la misma frecuencia, en este caso se les llama bimodal. Cuando existe

una distribución donde todos los valores tienen exactamente la misma frecuencia, se le llama distribución uniforme.

El cálculo de la moda en Python se hace por medio de las librerías scipy y statistics. Pandas, en su estructura de datos DataFrame, tiene métodos que permiten el cálculo de estas.

Cuando se usa scipy:

```
print("Moda según scipy:"+str(st.mode(diabete_Res["Glucose"])))
```

retorna:

Moda según scipy:ModeResult(mode=array([100]), count=array([17]))

Que significa que el valor con mayor frecuencia es 100 y que se repite 17 veces.

Cuando se usa el módulo statistics:

```
print("Moda según statistics:"+str(stat.mode(diabete_Res["Glucose"])))
```

El resultado es:

Moda según statistics:100

Cuando usamos pandas:

```
import pandas as pd

diabetes_df = pd.read_csv("diabetes.csv")

media = diabetes_df["Glucose"].mean()

mediana = diabetes_df["Glucose"].median()

moda = diabetes_df["Glucose"].mode()
```

Tema 2. Análisis exploratorio de datos y preprocesamiento

Medidas de dispersión

Estas medidas definen cuan agrupadas están las mediciones alrededor de una media o cuán lejos de ellas o dispersas. Por esto, también suelen llamárselas medidas de dispersión.

Entre estas se encuentran:

- Desviación.
- Varianza.
- Desviación estándar.
- Desviación Media Absoluta.
- Desviación Mediana absoluta desde la mediana.
- Rango.
- Percentil.
- Rango intercuartílico.

La intuición detrás de estas medidas de variabilidad es tratar de caracterizar la dispersión de los valores de un conjunto de datos de alguna de las medidas de tendencia centrar como la media o la mediana.

Para explicar este concepto, tomemos el conjunto de datos que usamos anteriormente: ,5,1,2,3.

Si tratáramos de encontrar una medida que caracterice este conjunto podríamos tomas las desviaciones alrededor de su media, por ejemplo:

5-2.75, 1-2.75, 2-2.75, 3-2.75.

Si sumamos todos los valores resultantes, ooooppppss, nos da cero, bueno pues no

es buena idea usar este procedimiento. Una mejor idea sería tomar el promedio de los valores absolutos de las desviaciones respecto a la media.

(2.25+ 1.75+0.75+0.25) / 4=1.25

A este tipo de procedimiento de caracterizar la desviación de los valores de un conjunto de datos se le llama **desviación media absoluta.**

Su fórmula matemática toma la siguiente forma:

$$Desviación\,media\,absoluta = \frac{\sum_{i=1}^{n} |(x_i - \bar{x})|}{n}$$

Esta medida es sumamente importante, pues si tomamos en vez de la media un valor conocido, esta medida podemos interpretarla como una medida de error y se llamaría Error Medio absoluto y nos ayudaría a caracterizar el error incurrido en la predicción de mediciones, por ejemplo. Esto lo veremos con mucho más detalle cuando estudiemos las métricas de evaluación de los modelos de aprendizaje automático.

Sin embargo, las más conocidas de las medidas de variabilidad son la varianza y la desviación estándar. A diferencia de la desviación absoluta, la varianza se basa en la desviación cuadrada, y su fórmula matemática es la siguiente:

$$S^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}$$

Mientras que la **desviación estándar,** es la raíz cuadrada de la varianza, y su fórmula es la siguiente:

$$S = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}}$$

Lo interesante de esta medida, la desviación estándar es que, es más fácil de interpretar, pues está expresada en la misma escala que los datos originales.

Tema 2. Análisis exploratorio de datos y preprocesamiento

Exploración de las distribuciones de frecuencia de un conjunto da datos

Histograma:

Uno de las maneras habituales para visualizar la distribución de frecuencia de un conjunto es por medio de un histograma.

Los histogramas se caracterizan por la cantidad de bin, divisiones o compartimentos tiene.

El eje x se divide en compartimentos y el eje y del gráfico muestra la cantidad de observaciones Veamos cómo se construye un histograma en Python.

Una de las formas más sencillas de mostrar un histograma en python es usando los métodos disponibles en la estructura DataFrame de Pandas, una de las librerías de Python para el análisis de datos.

```
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
#*************************************************
diabeteDF = pd.read_csv("diabetes.csv")
print("Media:"+str(diabeteDF["Glucose"].mean()))
print("Mediana:"+str(diabeteDF["Glucose"].median()))
print("Moda:"+str(diabeteDF["Glucose"].mode()))
diabeteDF["Glucose"].hist()
```

```
Media:120.89453125
Mediana:117.0
Moda:0     99
1    100
dtype: int64
<matplotlib.axes._subplots.AxesSubplot at 0x7fdc3ce9b990>
```

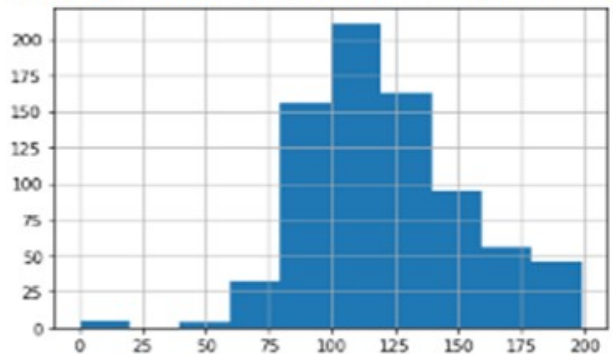

Figura 4. Histograma de un rasgo en un conjunto de datos. Fuente: elaboración propia.

¿Qué información podemos extraer de este tipo de gráfico? Pues a ojos ya podemos hacernos una idea de cómo están distribuidos los datos en este rasgo. De entrada podemos percatarnos que hay grupo de valores que están próximos al cero, si sabemos que los valores normales de glucosa en sangre son alrededor de 123, entonces nos damos cuenta que hay mediciones que contienen valores incorrectos. Es importante darse cuenta de los valores arrojados por la moda, que hay dos modas, una es 99 y la otra es 100.

En el eje x se ubican las concentraciones de glucosa en sangre de pacientes diabéticos o no. En el eje y se ubica el número de observaciones.

Veamos otra manera de construir un histograma en Python.

También puede usarse la librería matplotlib.pyplot para crear un histograma, especificando parámetros adicionales, como el número de bins o grupos en los que agrupar los datos.

```
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
#************************************************
diabeteDF = pd.read_csv("diabetes.csv")
print("Media:"+str(diabeteDF["Glucose"].mean()))
print("Mediana:"+str(diabeteDF["Glucose"].median()))
print("Moda:"+str(diabeteDF["Glucose"].mode()))
plt.hist(diabeteDF["Glucose"], bins=25)
```

```
Media:120.89453125
Mediana:117.0
Moda:0     99
1    100
dtype: int64
(array([ 5.,  0.,  0.,  0.,  0.,  1.,  0.,  5.,  9., 21., 48., 71., 85.,
        89., 74., 77., 57., 50., 42., 32., 26., 21., 23., 16., 16.]),
 array([  0.  ,   7.96,  15.92,  23.88,  31.84,  39.8 ,  47.76,  55.72,
         63.68,  71.64,  79.6 ,  87.56,  95.52, 103.48, 111.44, 119.4 ,
        127.36, 135.32, 143.28, 151.24, 159.2 , 167.16, 175.12, 183.08,
        191.04, 199.  ]),
 <a list of 25 Patch objects>)
```

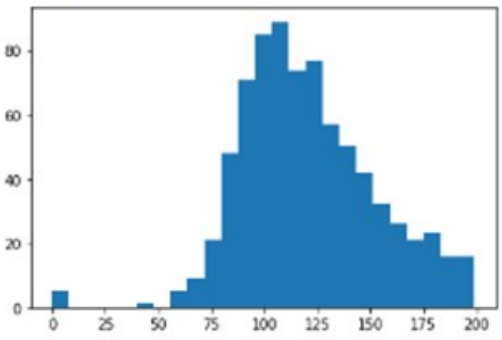

Figura 5. Ejemplo de histograma generado con pyplot, usando 25 bins.

Una conjunto de datos, entonces, puede quedar descrito por valores como la media, la mediana, la varianza y la desviación estándar. Sin embargo, los valores de este conjunto de datos pueden distribuirse de muchas maneras. Es a la forma en que se distribuyen a lo que se le llama, y valga la redundancia, distribuciones de frecuencias.

Los estadísticos, con el paso del tiempo, se han dado cuenta que los datos tomados, siempre tienen formas similares de distribuir sus valores y al ir sistematizando e investigando sus propiedades y características llegaron a generalizarlas para que así aparecieran las llamadas distribuciones teóricas de frecuencias.

Esa es la razón por la que la mayoría de las veces comparamos las distribuciones de nuestros datos con las distribuciones teóricas, pues de estas últimas ya conocemos sus propiedades y si nuestros datos actuales se ajustan a la forma de alguna de las teóricas podemos decir con alguna certeza que sus propiedades serán similares.

Ahora bien, ¿cuáles son las distribuciones de frecuencias más conocidas? Hay que saber, que si las variables que queremos comparar son continuas, las distribuciones a las que podemos comparar son a las llamadas distribuciones continuas:

- Distribución Normal.
- Distribución Ganma.
- Distribución Chi Cuadrado.
- Distribución t-Student.

Pero, también podemos listar algunas de las que se denominan distribuciones discretas:

- Distribución uniforme.
- Distribución Bernoulli.
- Distribución Geométrica.
- Distribución Hipergeométrica.
- Distribución Binomial.

En la Figura 6, se muestran la mayoría de las distribuciones más importantes y sus relaciones.

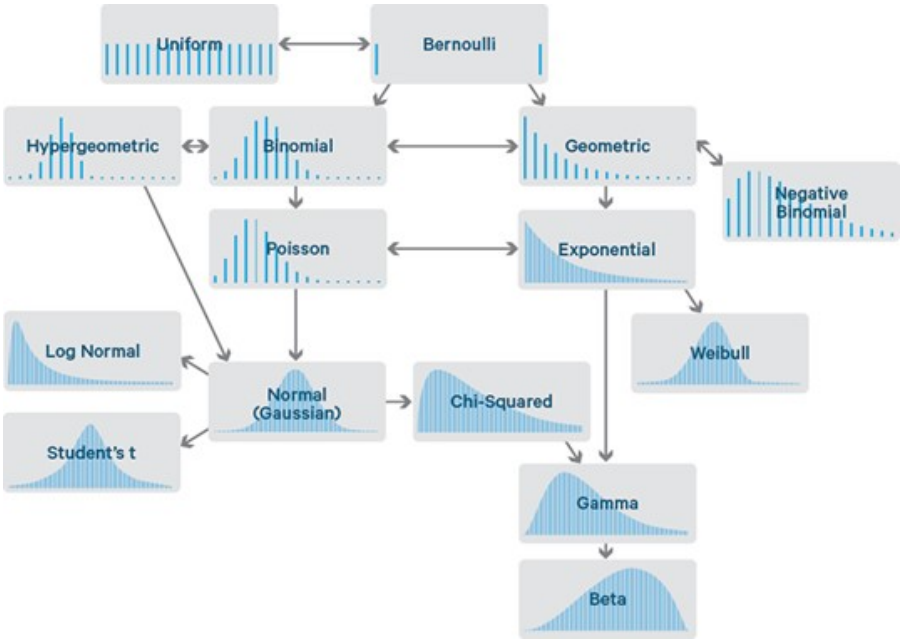

Figura 6. Distribuciones de frecuencias.

Distribución Normal: Se le conoce como distribución Gaussiana, distribución de Gauss, distribución campana, etc.

Esta distribución, al parecer, debe su popularidad a que muchos fenómenos naturales tienen distribuidos sus valores de esta forma.

Fenómenos como, las características morfológicas de los individuos, como la estatura y otros. Algunos efectos fisiológicos, como el efecto de los fármacos. El nivel de ruido en telecomunicaciones, etc.

Todos ellos tienen en común que sus valores están normalmente distribuidos. Las características principales de esta distribución son los siguientes:

- Es simétrica respecto a la media.
- En esta distribución, la media, la mediada y la moda son iguales.

- En el intervalo $[\mu-\sigma,\mu+\sigma]$ se encuentra, aproximadamente, el 68.26% de los valores de la distribución.
- En el intervalo $[\mu-2\sigma,\mu+2\sigma]$ se encuentra, aproximadamente, el 95,44% de los valores de la distribución.
- En el intervalo $[\mu-3\sigma,\mu+3\sigma]$ se encuentra, aproximadamente, el 99,74% de los valores de la distribución.
- Su expresión matemática es: $p\left(x;\mu,\sigma^2\right)=\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

Donde μ es la media aritmética σ^2es la desviación estándar.

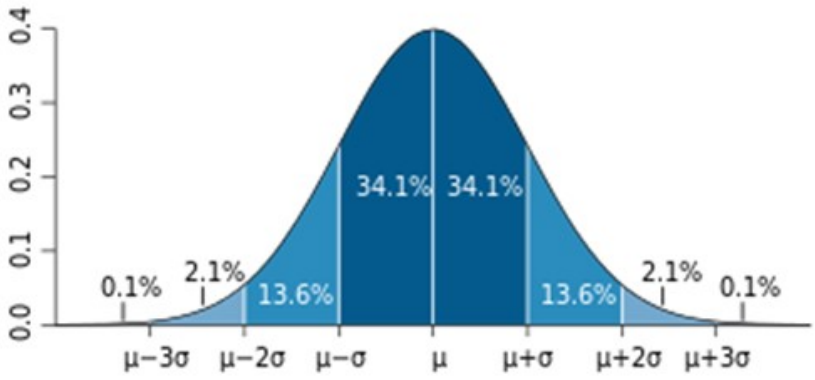

Figura 7. Intervalos en la gráfica de la distribución Gaussiana.

¿Por qué es tan importante la distribución normal? Esta pregunta aflora de manera natural en cuanto comenzamos a adentrarnos en el mundo del aprendizaje automático. Podríamos enumerar algunas, sin querer agotar el tema:

- Hay muchísimos fenómenos naturales que se pueden modelar por medio de esta distribución.
- Es más, según el teorema del límite central, cualquier fenómeno que puedan ser causados por un gran número de pequeños efectos, puede modelarse por esta distribución.

- Tiene propiedades matemáticas muy bien conocidas y explotadas en la inferencia estadística y otras ramas.
- Muchos métodos de aprendizaje automático, incluyendo Análisis de Discriminante lineal y Regresión asumen que los datos de entrenamiento están normalmente distribuidos.
- Algunos modelos de redes neuronales, necesitan datos normalizados, para su entrenamiento.

Medidas de forma de las distribuciones

Conociendo que los conjuntos de datos, o rasgos, se pueden representar o caracterizar por la distribución de la frecuencia de sus valores, estamos en condiciones de aprender sobre algunas medidas que expresan la forma de esta distribución.

De estas medidas veremos la asimetría, que explica si la forma de la distribución está estirada a la derecha o a la izquierda, es decir es asimétrica.

Este tipo de comportamiento de la distribución de los valores de un rasgo es automáticamente reconocido por medio del histograma pero también medidas que lo sumarizan de manera numérica.

Asimetría *(Skewness)*

Es la medida que indica la simetría de la distribución de una variable respecto a la media aritmética, sin necesidad de hacer la representación gráfica. Los coeficientes de asimetría indican si contiene el mismo número de elementos a izquierda y derecha de la media.

La asimetría arroja valores para identificar la forma de la distribución, que puede ser:

- **Asimetría negativa:** la cola de la distribución se alarga para valores inferiores a la media.
- **Simétrica:** contiene el mismo número de elementos a izquierda y derecha de la media. En este caso, coinciden la media, la mediana y la moda. La distribución se adapta a la forma de la campana de Gauss, o distribución normal.
- **Asimetría positiva:** la cola de la distribución se alarga (a la derecha) para valores superiores a la media.
- En la Figura 8 se muestras las formas respectivas.

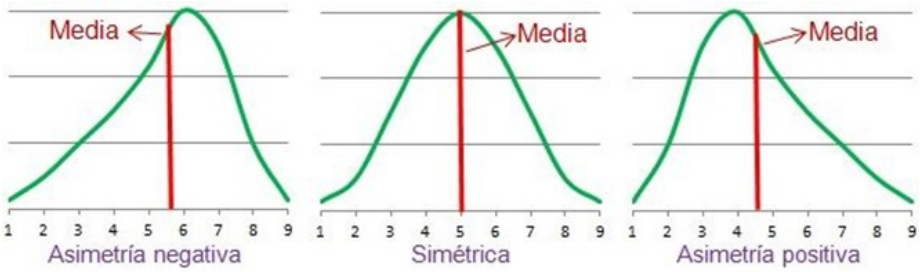

Figura 8. Tipos de asimetría.

Para calcular la asimetría de una variable, hay principalmente tres métodos:

- Coeficiente de asimetría de Fisher.
- Coeficiente de asimetría de Pearson.
- Coeficiente de asimetría de Bowley.

Coeficiente de asimetría de Fisher:

El coeficiente de asimetría de Fisher CAF evalúa la proximidad de los datos a su media x. Cuanto mayor sea la suma ∑(xi–x)3, mayor será la asimetría. Sea el conjunto X=(x1, x2,…, xN), entonces la fórmula de la asimetría de Fisher es:

$$CA_F = \frac{\sum_{i=1}^{N}(x_i - \overline{x})^3}{N \cdot S_x^3}$$

siendo $\overline{x}$ la media y S_x la desviación típica

Y lo más importante es cómo analizamos los resultados que nos arroja esta correlación:

- Si CAF<0: la distribución tiene una asimetría negativa y se alarga a valores menores que la media.
- Si CAF=0: la distribución es simétrica.
- Si CAF>0: la distribución tiene una asimetría positiva y se alarga a valores mayores que la media.

Esto significa que en la medida que el coeficiente de asimetría esté más cerca de cero, la distribución de los datos analizados está más cerca también de ser normalmente distribuidos.

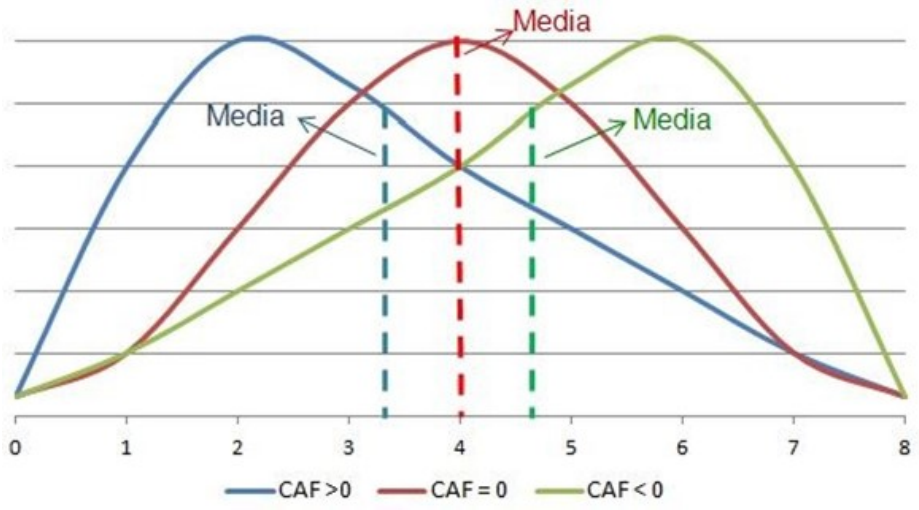

Figura 9. Resultados del coeficiente de asimetría de Fisher.

Coeficiente de asimetría de Pearson:

El coeficiente de asimetría de Pearson CAP mide la diferencia entre la media y la moda con respecto a la dispersión del conjunto X=(x1, x2,…, xN).

Este procedimiento, se emplea solamente para distribuciones unimodales y poco asimétricas. Su formulación matemática es:

$$CA_P = \frac{\overline{x} - Mo(X)}{S_x}$$

siendo $\overline{x}$ la media, $Mo(X)$ la moda y S_x la desviación típica

Este tipo de coeficiente explota la característica de las distribuciones normales, en las cuales la media y la moda son iguales. Entonces tenemos que:

- Si CAP<0: la distribución tiene una asimetría negativa, puesto que la media es menor que la moda.
- Si CAP=0: la distribución es simétrica. Media y Moda coinciden.