Clasificación desde la Econometría

¡Comencemos!

¿Qué aprenderás?

- Conocer la regresión logística y sus fundamentos.
- Reconocer los supuestos en que tiene sustento teórico.
- Implementar un modelo de regresión con `statsmodels`.
- Implementar un modelo predictivo con `scikit-learn`.

Introducción

En esta semana estudiaremos el problema de clasificación en Data Science, que busca esclarecer sobre los efectos que tienen los atributos en una variable categórica.

Siguiendo la taxonomía de Hastie, Tibshinari y Friedman (2009), los problemas de clasificación corresponden a un ejemplo de aprendizaje supervisado donde el vector objetivo responde a un atributo discreto. Ejemplos de ello:

- Movimientos del mercado: ¿Bajará o subirá la bolsa? $\rightsquigarrow Y_i \in \{0: Baja, 1: Sube\}$.
- Clasificación de Spam: ¿Es este mail Spam o No? $p - value \leq .025\ Y_i \in 0: No, 1: S.$
- Optimización de Preferencias: ¿Es más probable votar o no?
  $Y_i \in 0: NoVota, 1: Vota$

¡Vamos con todo!

Los ejemplos mencionados hacen referencia a un problema de clasificación binario, donde observamos la presencia o ausencia de un atributo. La aproximación de este fenómeno toma forma en un ensayo de Bernoulli.

```
# Importamos la triada clásica
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
# seaborn
import seaborn as sns
# scipy stats para simular
import scipy.stats as stats
# statsmodels para modelación
import statsmodels.api as sm
import statsmodels.formula.api as smf

import warnings
warnings.filterwarnings("ignore")

import lec6_graphs as gfx

plt.style.use('seaborn') # Gráficos estilo seaborn
# plt.rcParams["figure.figsize"] = (6, 3) # Tamaño gráficos
# plt.rcParams["figure.dpi"] = 200 # resolución gráficos
```

Estimando probabilidades de éxito y fracaso

La tarea de esta lectura es el uso de pozos de agua en Bangladesh. Resulta que el agua en Bangladesh y otros países del Sudeste Asiático está contaminada con arsénico natural. El riesgo del arsénico como cancerígeno y catalizador de otras enfermedades aumenta con la exposición prolongada. El gobierno de Bangladesh inició una estrategia nacional para la mitigación de riesgos asociados al arsénico. Una de las estrategias era el incentivar el uso compartido de pozos de napa subterránea, dado que presentan tasas de arsénico sustancialmente menores.

Usaremos una base de datos donde se registró si un hogar comenzó a utilizar un nuevo pozo de napa subterránea o no. Así, clasificaremos como 1 si el hogar se cambió a un pozo nuevo, 0 de lo contrario. Analizaremos la probabilidad de cambiarse o no en función a las siguientes variables:

1. La distancia del hogar al pozo más cercano (medida en 100 metros) (`dist100`).
2. El nivel de arsénico del actual pozo en uso del hogar (`arsenic`).
3. El nivel educacional de el/la jefe/a de hogar (`educ4`).
4. Si algún miembro de la familia participa o no en asociaciones comunitarias (`assoc`).

Importamos la base de datos con `pandas`.

```
# ingresamos la base de datos
df = pd.read_csv('wells.csv')
# La base de datos incluye una columna de índice. Eliminemosla para
evitar futuros conflictos
df = df.drop("index", axis =1)
```

Análisis exploratorio

La base de datos se compone de 3020 observaciones y cinco columnas.

```
print("La base de datos tiene ", df.shape[0], "observaciones y ",
df.shape[1], " columnas")
print("Las variables de la base de datos son ", df.columns)
```

```
La base de datos tiene  3020 observaciones y  5  columnas
Las variables de la base de datos son  Index(['y', 'dist100', 'arsenic',
'educ4', 'assoc'], dtype='object')
```

Solicitemos las medidas descriptivas de los datos con los que vamos a trabajar. Como ya sabemos que el método `describe` de pandas funciona sólo con las columnas numéricas, utilizaremos `value_counts` para las columnas con atributos discretos.

Esto lo implementamos en un `for` con un `if` para diferenciar si los atributos tiene más de dos niveles.

```
for i in df:
    if len(df[i].value_counts()) > 2:
        print(df[i].describe(), "\n")
    else:
        print(df[i].value_counts('%'), "\n")
```

```
1    0.575166
0    0.424834
Name: y, dtype: float64

count    3020.000000
mean        0.483319
std         0.384787
min         0.003870
25%         0.211172
50%         0.367615
75%         0.640410
max         3.395310
Name: dist100, dtype: float64
```

```
count    3020.000000
mean        1.656930
std         1.107387
min         0.510000
25%         0.820000
50%         1.300000
75%         2.200000
max         9.650000
Name: arsenic, dtype: float64

count    3020.000000
mean        1.207119
std         1.004329
min         0.000000
25%         0.000000
50%         1.250000
75%         2.000000
max         4.250000
Name: educ4, dtype: float64

0    0.577152
1    0.422848
Name: assoc, dtype: float64
```

```
df.describe()
```

	y	dist100	arsenic	educ4	assoc
count	3020.000000	3020.000000	3020.000000	3020.000000	3020.000000
mean	0.575166	0.483319	1.656930	1.207119	0.422848
std	0.494400	0.384787	1.107387	1.004329	0.494093
min	0.000000	0.003870	0.510000	0.000000	0.000000
25%	0.000000	0.211172	0.820000	0.000000	0.000000
50%	1.000000	0.367615	1.300000	1.250000	0.000000
75%	1.000000	0.640410	2.200000	2.000000	1.000000
max	1.000000	3.395310	9.650000	4.250000	1.000000

Imagen 1. Resultado 1.
Fuente: Desafío Latam.

Respecto a las variables con atributos discretos (nuestra variable dependiente `y`, y `assoc`), observamos que alrededor del 58% de las familias componentes de la muestra se cambiaron de pozo a uno más seguro. Encontramos lo opuesto con `assoc`, cerca del 58% de los jefe de hogar en la familia no participan en asociaciones comunitarias. El nivel educacional de los encuestados se mide en el máximo grado académico desde 0 (Sin educación formal) a 4 (Secundaria completa). La media reportada sugiere que gran parte de la población posee niveles bajos de educación.

La mayoría de las familias se sitúan en una distancia de 48 metros de un pozo seguro, y el nivel de arsénico promedio en el agua es de 1.65, entre el rango de .51 y 9.65, lo cual es relativamente bajo.

Para tener una mejor perspectiva del comportamiento de las columnas, vamos a graficarlas con histogramas para las continuas y gráficos de barra para las discretas.

```
for n, i in enumerate(df):
    plt.subplot(2, 3, n+1)
    if len(df[i].value_counts()) > 2:
        sns.distplot(df[i])
        plt.title(i)
        plt.xlabel("")
    else:
        sns.countplot(y=df[i])
        plt.title(i)
        plt.xlabel("")
    plt.tight_layout()
```

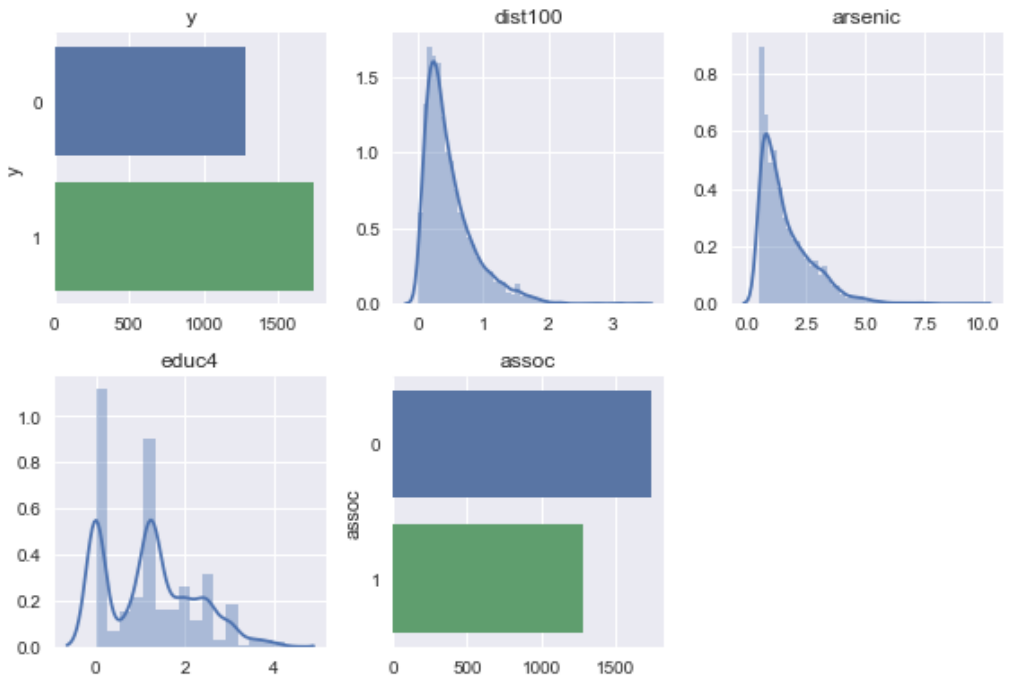

Imagen 2. Histogramas y gráficos.
Fuente: Desafío Latam.

Observamos que para `dist100`, `arsenic` y `educ4` existe un sesgo hacia los valores bajos, lo cual sugiere que los casos donde la distancia y el nivel de arsénico son sustancialmente grandes son relativamente anómalos.

Respecto a las variables discretas, observamos que las clases están relativamente bien equilibradas, lo cual facilita nuestra modelación.

Digresión: ¿Qué hacer ante clases desbalanceadas?

Existe una serie de mecanismos alternativos para mitigar el efecto del desbalance en las clases:

1. Asumir que el componente estocástico del modelo se puede representar de mejor manera mediante una distribución Poisson, Binomial Negativa o Cero Dispersa.
2. Utilizar errores robustos (Huber-White sandwich) para fortalecer la matriz de varianza-covarianza.
3. Estimar el modelo con robit (regresión logística t-distribuída) por sobre logit.
4. Recolectar más datos en la medida que sea posible.
5. Remuestrear sobre la muestra para disminuir el desbalance:
    - **Over-sampling:** Remuestrear copias de la clase sub-representada a la muestra original.

- **Under-sampling:** Eliminar instancias de la clase sobrerrepresentada.

Ahora visualizamos las correlaciones mediante una matriz. Seaborn presenta el método `sns.heatmap()` para generar un gráfico sobre la intensidad de las asociaciones entre las variables de una base de datos. Los parámetros ingresados son:

1. `df.corr()` ingresamos las correlaciones calculadas mediante `pandas`.
2. `cmap='Blues'` define el rango de colores. En este caso utilizaremos una gama de azules.
3. `annot=True` añadirá el puntaje de la correlación sobre cada celda.

```
sns.heatmap(df.corr(), cmap='Blues', annot=True);
```

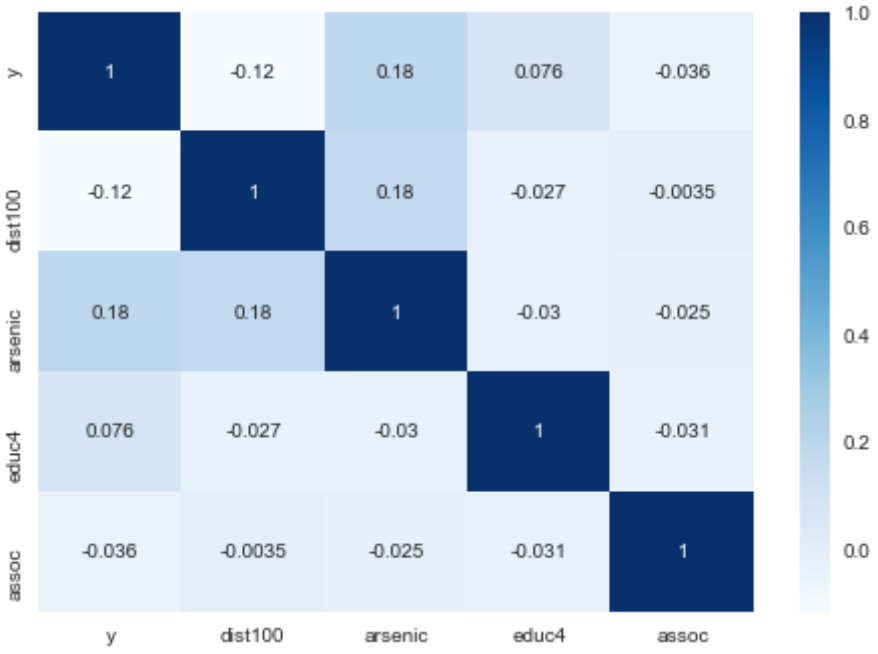

Imagen 3. Respuesta *sns.heatmap()*.

Los resultados de la matriz de correlación pueden ser desalentadores, dado que no hay asociaciones fuertes entre las variables. Por defecto utilizamos la correlación de pearson, que no tiene un buen desempeño con variables que no sean estrictamente normales. Como ya se ha mencionado antes, esto no es un impedimento para seguir con la modelación ya que muchas relaciones no son estrictamente lineales.

El modelo de probabilidad lineal

Una primera aproximación al problema de clasificación binario es utilizar una regresión lineal, asumiendo que en nuestra variable dependiente estaremos midiendo la probabilidad del suceso.

$$\mathbf{y}_i = \beta_0 + \beta_1 \times \mathbf{dist100} + \varepsilon_i$$

Este modelo se conoce como modelo de probabilidad lineal, donde implementamos una regresión lineal con nuestra variable dependiente binaria. Los parámetros estimados afectan la probabilidad de que el evento suceda. Con nuestro $\widehat{Y}$ predicho podemos clasificar una observación como 1 si $\widehat{Y} > 0.5$, de lo contrario 0.

```
# ejecutemos nuestro modelo
m1_ols = smf.ols('y ~ dist100', df).fit()
```

Ahora vamos a generar una función para obtener sólo los parámetros necesarios en nuestro modelo. La función `concise_summary()` necesita como argumento el modelo estimado con `statsmodels` y devolverá sólo algunas características esenciales del modelo.

```
def concise_summary(mod, print_fit=True):
    #guardamos los parámetros asociados a estadísticas de ajuste
    fit = pd.DataFrame({'Statistics': mod.summary2().tables[0][2][2:],
                        'Value': mod.summary2().tables[0][3][2:]})
    # guardamos los parámetros estimados por cada regresor.
    estimates = pd.DataFrame(mod.summary2().tables[1].loc[:, 'Coef.':
'Std.Err.'])
    # imprimir fit es opcional
    if print_fit is True:
        print("\nGoodness of Fit statistics\n", fit)

    print("\nPoint Estimates\n\n", estimates)

# solicitemos las características del modelo
concise_summary(m1_ols)
```

```
Goodness of Fit statistics
             Statistics      Value
2                  BIC:  4288.4686
3      Log-Likelihood:    -2136.2
4          F-statistic:      42.57
5  Prob (F-statistic):   7.95e-11
6                Scale:    0.24111

Point Estimates

               Coef.  Std.Err.
Intercept  0.648407  0.014347
dist100   -0.151539  0.023225
```

Una de las primeras preguntas que nos hacemos es **¿dónde está nuestro R-cuadrado?** Resulta que no es tan relevante para nuestro modelo, dado que mide simplemente la capacidad explicativa de la variabilidad de ***Y*** dado un conjunto de variables explicativas. Preferimos concentrarnos en aspectos menos triviales como si por lo menos uno de nuestros parámetros es distinto a cero, que se mide con `Prob (F-statistic)`. Aspectos como el BIC (Bayesian Information Criteria) y el Log-Likelihood son más informativos sobre la optimización de nuestro método que el R-cuadrado.

Sobre métricas de bondad de ajuste en clasificación

Detengámonos un momento a tratar el tema de medir la bondad de ajuste de nuestro modelo de clasificación a los datos, para esto, veamos los estadísticos que aparecen en la tabla anterior uno a uno:

1. BIC (Bayesian Information Criteria - Criterio de Información Bayesiano):

Respecto a los puntos estimados, nuestra función reporta sólo el coeficiente y su error estándar asociado. La interpretación de los coeficientes en este modelo son:

- El intercepto (β_o) se puede interpretar al considerar `dist100=0`. Una familia que esté a 0 metros de distancia de un pozo tendrá una probabilidad del 65% de cambiarse de pozo (asumiendo que está limpio).

- El tema es que la función `concise_summary` no devuelve nuestra significancia estadística. Para obtener el puntaje z, simplemente dividimos el parámetro estimado por su desviación estándar. En este caso ***z= 0.648 /0.014 = 46.28***. Al ser superior a los puntajes de corte, existe evidencia para rechazar la hipótesis nula al 95%.

- `dist100` (β_1) se puede interpretar como la diferencia entre dos individuos que tienen similares características pero difieren en 100 metros de distancia de un pozo seguro, conlleva a una disminución de un 15% en la probabilidad de cambiarse de pozo. El parámetro es significativo con un z de **z = -0.151 /0.023 = -6.56** es superior a los puntos críticos por lo que existe evidencia para rechazar la hipótesis nula.

```
sns.lmplot('dist100','y', df,
           line_kws={'color':'tomato'},
           scatter_kws={'color': 'grey', 'alpha': .5});
```

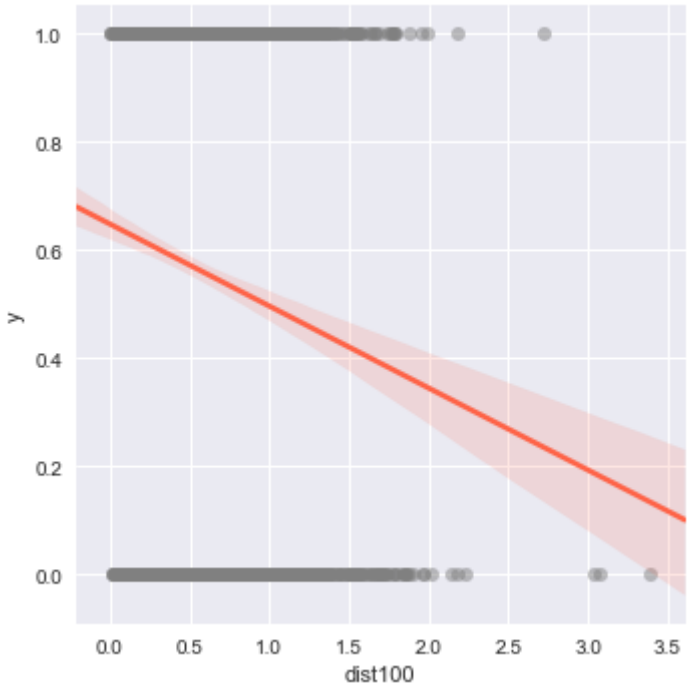

Imagen 4. Resultado de *sns.lmplot()*.
Fuente: Desafío Latam.

Podemos visualizar la relación entre ambas variables con `sns.lmplot`, que solicita el eje x `dist100` y la dependiente `y`. Los argumentos `line_kws` y `scatter_kws` gobiernan los elementos visuales de los puntos y la recta. De forma intuitiva se demuestra que en la medida que la probabilidad de cambiarse de pozo disminuye en la medida que la familia se encuentra más alejada de un pozo.

Una de las ventajas del modelo de probabilidad lineal es la facilidad de su interpretación, dado que los coeficientes hacen referencia al cambio en una unidad de X afecta a la probabilidad ***Pr(y=1)***. Lamentablemente, no está exento de problemas la probabilidad estimada por este modelo puede tomar valores mayores que 1 y menores que 0, lo cual va más allá de los límites razonables de la probabilidad.

Regresión Logística al rescate

Si bien nuestro estimador LPM es intuitivo, presenta fallas severas en la estimación. La regresión logística permite generar estimaciones mediante el método de máxima verosimilitud (se elige el modelo tal que su verosimilitud sobre los datos es la máxima) y toma en consideración los problemas anteriormente vistos. Los detalles sobre el método de máxima verosimilitud se los dejaremos a Python :).

Resulta que nuestro objetivo es desarrollar un modelo predictivo para la probabilidad de ocurrencia de Y ($Pr(x) = Pr(Y = 1|X = x)$).

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 \times \texttt{dist100}$$

Implementar la regresión logística con `statsmodels` es similar a la implementación del modelo de regresión lineal. Necesitamos declarar la ecuación como un string y definir el objeto donde `statsmodels` buscará las variables.

El método requiere que nuestra variable dependiente sea binaria. Los regresores independientes pueden ser continuos y/o categóricos.

```
m1_logit = smf.logit('y ~ dist100', df).fit()
concise_summary(m1_logit)
```

```
Optimization terminated successfully.
         Current function value: 0.674874
         Iterations 4

Goodness of Fit statistics
         Statistics      Value
2             AIC:  4080.2378
3             BIC:  4092.2639
4  Log-Likelihood:    -2038.1
5         LL-Null:    -2059.0
6           Scale:     1.0000

Point Estimates
              Coef.  Std.Err.
Intercept  0.605959  0.060310
dist100   -0.621882  0.097426
```

¿Y qué significa todo esto para nosotros?

Podríamos estar tentados a interpretar los coeficientes como: "La diferencia entre dos individuos que tienen similares características pero *difieren en 100 metros de distancia de un pozo seguro*, conlleva a una disminución de un .62 en la probabilidad de cambiarse de pozo.

Esto no es correcto, dado que los parámetros estimados mediante la regresión logística se conocen como *log-odds*, que representan el logaritmo de la oportunidad (chance) de ocurrencia de un evento en específico. Esto obliga a tomar con cautela el cómo se deben interpretar. La interpretación de arriba tiene sentido cuando hablamos del *logaritmo de la oportunidad de ocurrencia*, lo cual resulta difícil de entender.

La regresión logística no está exenta de problemas y presunciones:

1. Perdemos en gran medida la capacidad interpretativa del modelo, por ejemplo, nos podría interesar estudiar el efecto de cada regresor sobre la probabilidad de ocurrencia de las clases en la variable dependiente, sin embargo, al estar estos coeficientes usándose para cuantificar el efecto de las variables independientes sobre logaritmo de las posibilidades (log-odds, la transformación de y por la función de link). Es más difícil, entonces, observar el efecto que tiene una variable independiente en la probabilidad de los eventos de y observando simplemente los valores de los coeficientes.

2. Por la forma en la que es ajustado el modelo de probabilidad (máxima verosimilitud) tener clases desbalanceadas (gran parte de la muestra pertenece a una sola clase y muy pocos elementos a la segunda clase) hace que el modelo ajustado se vea altamente influenciado por la gran cantidad de elementos de la clase mayoritaria durante el ajuste por verosimilitud.

3. Es sensible, al igual que un modelo de regresión lineal, a valores extremos en los predictores continuos.

4. Se asume linealidad entre los log-odds (log-posibilidades) de y y cada variable predictora, esto implica monotonicidad en esta relación y por lo tanto no contempla los casos en los que esta relación es de un orden superior. Esto se puede ver en la forma funcional del modelo donde modelamos la transformación de y mediante la función de link como una combinación lineal de las variables predictoras.

5. Requiere independencia entre los ejemplos de la muestra, es decir, los ejemplos en la matriz X no pueden venir de remuestreos donde se dupliquen los ejemplos.

Digresión: Oportunidad de ocurrencia es distinta a probabilidad.

Aunque en el ámbito cotidiano se utiliza '*oportunidad de ocurrencia*' (Odds) y *'probabilidad'*(probability) como sinónimos, en estadística (y en especial en teoría de juegos) estos dos conceptos son distintos, aunque están fuertemente relacionados entre sí:

La probabilidad de que ocurra un suceso se define (clásicamente) de manera empírica, es decir:

$$p(x) = \frac{\text{N}^{\text{o}} \text{ de ocurrencias de x}}{\text{Casos totales medidos}}$$

La oportunidad de que ocurra ese mismo suceso es:

$$Odd(x) = \frac{p(x)}{1 - p(x)}$$

La diferencia entre **oportunidad de ocurrencia y probabilidad** es que la oportunidad de ocurrencia nos dice qué tanto del espacio de eventos posibles está siendo utilizado por el evento de interés, mientras que la probabilidad nos dice qué tan "seguros" podemos estar el resultado final. Por ejemplo, imaginemos que vamos a una casa de apuestas de caballos y nos enteramos que al caballo '*Lucky Luke*' ha ganado 30 de las 100 carreras que ha corrido, luego, la probabilidad de que 'Lucky Luke' gane la siguiente carrera es de ***30/100 = 0.30***, mientras que las oportunidades de que gane son de ***0.30/(1-0.30) = 0.43***, o dicho más cotidianamente, 3 victorias a 7 derrotas.
Finalmente, notar que las oportunidades de ganar pueden ser un número en todo el intervalo [*0, ∞ +*], lo cuál es justo lo que necesitamos para poder utilizar la regresión lineal en este caso donde nuestra recta de separación entre las clases debe poder extenderse más allá del rango [*0, 1*] que nos entrega la medida de probabilidad.

Esta última relación matemática nos da la razón de existencias de ese logaritmo en la expresión presentada al inicio de esta sección: **Aplicar la función *´logit´* a la probabilidad nos entrega el logaritmo de las oportunidades (odds)**.

El objetivo es traducir los valores del log odds en una declaración de probabilidad entre 0 y 1. Así se genera una explicación intuitiva sobre el efecto de una variable en la probabilidad de ocurrencia (por sobre el efecto de una variable en el logaritmo de la oportunidad de ocurrencia). Para ello utilizamos la función logística inversa (presentada como $logit\ -\ 1(x)\ = \frac{exp(x)}{1+exp(-x)}$).

Supongamos que deseamos saber la probabilidad de cambiarse a un pozo seguro para toda la muestra. En base a nuestro modelo estimado, podemos obtener la probabilidad con los siguientes pasos:

1. Calcular el log odds promedio cuando dist100 es igual al promedio

Primero debemos reemplazar valores en nuestra ecuación para interpolar el punto predicho cuando `dist100` toma el promedio:

$$\Pr(\text{CambioPozo} = 1|X) = \log\left(\frac{exp(\beta_0 + \beta_1)}{1 - exp(\beta_0 + \beta_1)}\right)$$

Esto lo podemos aplicar de la siguiente manera:

```
# guardamos la media en un objeto
dist100_mean = df['dist100'].mean()
print("La media es de ", round(dist100_mean, 2))

# accedemos a los parámetros con la sintáxis modelo.params['parametro]
estimate_y = m1_logit.params['Intercept'] + (m1_logit.params['dist100']
* dist100_mean)
print("El log odds estimado es de ", round(estimate_y, 2))
```

```
La media es de  0.48
El log odds estimado es de  0.31
```

2. Convertir nuestro log odds estimado en una probabilidad

Este punto estimado debemos traducirlo a una probabilidad por medio de la función logística inversa, que definimos en Python como una función:

```
def invlogit(x):
    return 1 / (1+np.exp(-x))

print("La probabilidad promedio de cambiarse de pozo cuando tenemos una
distancia de 48 metros es: ",
      round(invlogit(estimate_y), 2))
```

```
La probabilidad promedio de cambiarse de pozo cuando tenemos una
distancia de 48 metros es:  0.58
```

Realizar el mapeo de log odds a probabilidades puede ser tedioso, pero le damos mucho más sentido a nuestro modelo.

Calculando el efecto diferencial

También podemos estar interesados en el cambio en la probabilidad cuando nuestro ***x*** aumenta en 1 unidad. Esto se asemeja a nuestra conceptualización clásica del coeficiente estimado en la regresión lineal.

De manera similar a como lo hicimos con el efecto promedio, debemos generar las funciones logísticas inversas para estimar la probabilidad de dos eventos, mediante los cuales simularemos el cambio en la probabilidad cuando nuestro ***x*** cambia en una unidad.

Vamos a generar cuatro escenarios donde se estimará la probabilidad de cambio para 100, 200, 300 y 400 metros.

```
pr_dist_100 = invlogit(m1_logit.params['Intercept'] +
(m1_logit.params['dist100'] * 1))
pr_dist_200 = invlogit(m1_logit.params['Intercept'] +
(m1_logit.params['dist100'] * 2))
pr_dist_300 = invlogit(m1_logit.params['Intercept'] +
(m1_logit.params['dist100'] * 3))
pr_dist_400 = invlogit(m1_logit.params['Intercept'] +
(m1_logit.params['dist100'] * 4))
```

Si deseamos ver cuál es el cambio de la probabilidad en una unidad, restamos dos unidades estimadas. Las líneas de abajo:

```
print("La probabilidad de cambiar de pozo entre 100 y 200 metros: ",
round(pr_dist_100 - pr_dist_200, 3))
print("La probabilidad de cambiar de pozo entre 200 y 300 metros: ",
round(pr_dist_200 - pr_dist_300, 3))
print("La probabilidad de cambiar de pozo entre 300 y 400 metros: ",
round(pr_dist_300 - pr_dist_400, 3))
```

```
La probabilidad de cambiar de pozo entre 100 y 200 metros:  0.15
La probabilidad de cambiar de pozo entre 200 y 300 metros:  0.125
La probabilidad de cambiar de pozo entre 300 y 400 metros:  0.089
```

Se observa que el tránsito entre distintos valores conlleva a distintos cambios en las probabilidades. Este es uno de los principales motivos de esta cautela en la interpretación del modelo logístico: **No se puede asumir monotonicidad estricta en los parámetros**. La probabilidad de un cambio entre 200 y 300 (***Pr(x) = .124***) es distinta a la probabilidad entre 300 y 400 (***Pr(x)= .088***).

Podemos visualizar la estimación logística mediante el método `sns.lmplot`, incluyendo un argumento `logistic=True` que estimará la recta de ajuste en función a los log-odds.

La línea azul vertical se conoce como el límite de decisión, situación hipotética donde una observación tiene iguales oportunidades del evento en cuestion.

En este este caso, cuando una familia se encuentre aproximadamente a 97 metros de un pozo seguro, tendrán iguales oportunidades de cambiarse o no. Para estimar el límite de decisión, se utiliza la siguiente fórmula:

$$x_1 = \frac{-\hat{\beta}_0}{\hat{\beta}_1}.$$

```
sns.lmplot('dist100', 'y', df,
           logistic=True,
           line_kws={'color':'tomato', 'lw': 3},
           scatter_kws={'color': 'lightgrey', 'alpha': .5})
```

```
decision_boundary = - m1_logit.params['Intercept'] /
m1_logit.params['dist100']
plt.axvline(decision_boundary, lw=1, color='dodgerblue')
plt.axhline(.5, linestyle='--', color='grey', lw=1)
plt.text(decision_boundary+.2, .5+.05, r'Caso donde $\hat{p}=.5$',
color='forestgreen')
plt.plot(decision_boundary, .5, 'o', color='forestgreen')


print("Una observación tiene igual probabilidad en ambos sucesos cuando
x = ", round(decision_boundary, 3));
```

```
Una observación tiene igual probabilidad en ambos sucesos cuando x =
0.974
```

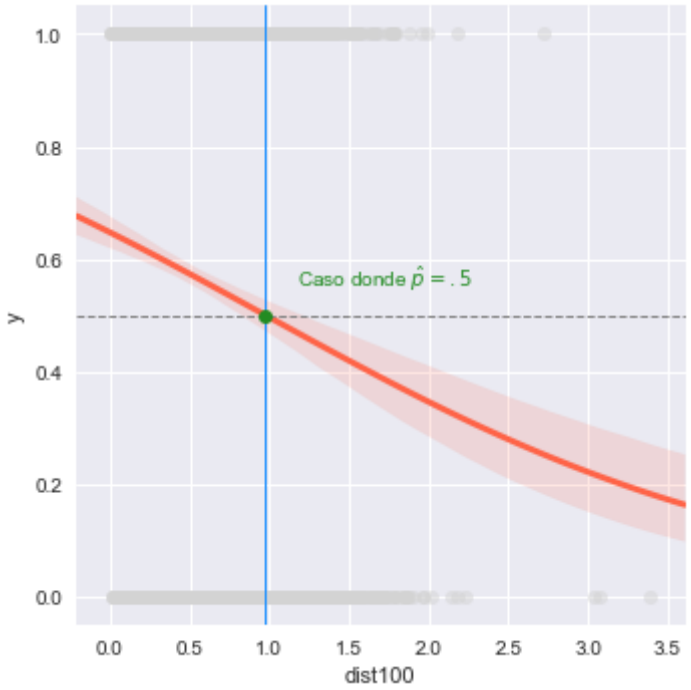

Imagen 5. Resultado efecto diferencial.
Fuente: Desafío Latam.

Relación entre LPM y Logit

Dado que sabemos la idoneidad del modelo logístico para los casos binarios, ¿Por qué utilizamos el modelo de probabilidad lineal?

Resulta que podemos tomar los log-odds de un modelo logístico y dividirlos por cuatro para obtener un intervalo superior de la contribución de x en y cuando cambia en una medida. Este punto es una aproximación al comportamiento estimado en el medio de la curva logística donde las probabilidades son cercanas al .5.

Digresión: Aspectos formales de la regla dividir en 4

Resulta que la curva logística es más pronunciada en el centro, donde $\beta 0 + \beta 1x = 0$, y su función logística inversa es $logit - 1(\beta 0 + \beta 1x) = 0.5$.

La derivada de la función lineal se maximiza en este punto con un valor de $\beta 1exp(0)/(1 + exp(0))2 = \beta 1/4$.
Así $\beta 1/4$es la diferencia máxima de $Pr(y = 1|x)$.

Tomemos los coeficientes de nuestro modelo $y = \beta 0 + \beta 1 \times dist100$ para ambos modelos.

```
print("\nOLS")
concise_summary(m1_ols, print_fit=False)
print("\nLogit")
concise_summary(m1_logit, print_fit=False)
```

```
OLS

Point Estimates

              Coef.  Std.Err.
Intercept  0.648407  0.014347
dist100   -0.151539  0.023225

Logit
Point Estimates

              Coef.  Std.Err.
Intercept  0.605959  0.060310
dist100   -0.621882  0.097426
```

Si dividimos el coeficiente estimado de nuestra regresión logística $\$ -.62 / 4 = .15\$$, veremos que es una aproximación razonable del coeficiente estimado en nuestro modelo LPM.

Este coeficiente corresponde a la máxima diferencia contribuída por x en $Pr(y = 1|x)$. Si revisamos nuestros efectos estimados, observaremos que el .15 estimado corresponde a la diferencia en un 15% de probabilidad entre una familia que está a 100 metros de distancia y otra que está a 200 metros de distancia de un pozo seguro.

```
print("La probabilidad de cambiar de pozo entre 100 y 200 metros: ",
round(pr_dist_100 - pr_dist_200, 3))
```

```
La probabilidad de cambiar de pozo entre 100 y 200 metros:  0.15
```

El problema de clasificación desde Machine Learning

El objetivo de la clasificación es enseñarle a la máquina a discriminar entre un número finito de clases en base a una serie de atributos.

Para aproximarnos a este problema diseñamos una serie de aproximaciones funcionales candidatas para facilitar la discriminación entre las clases.

El objetivo es realizar predicciones en nuevas observaciones en base a la función candidata que presente el mejor desempeño predictivo.

Mientras que en los problemas de regresión el desempeño se medía mediante la reducción del error cuadrático, en la clasificación se busca aumentar la tasa de clases predichas correctamente y reducir los falsos positivos (situaciones donde se clasifica de forma errónea) y falsos negativos (situaciones donde el clasificador ignora clasificaciones exitosas).

Implementando un modelo logístico con sklearn

Utilizaremos la clase `LogisticRegression` dentro del módulo `linear_model` de `sklearn`. Como todo modelo candidato de `sklearn`, seguimos los pasos de inicializarlo al ingresar datos y posteriormente ejecutarlo.

Comencemos por segmentar nuestra base de datos en entrenamiento y pruebas siguiendo la nomenclatura clásica, guardando el 33% de la muestra para pruebas. Recuerden especificar una semilla pseudoaleatoria con `random_state` para asegurar replicabilidad de los resultados.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.loc[:,
'dist100':'assoc'],
                                                    df['y'],
                                                    test_size=.33,
                                                    random_state=11238)
```

De forma adicional estandarizamos cada atributo en nuestras muestras mediante `StandardScaler` en el módulo `sklearn.preprocessing`. Mediante la estandarización transformamos las variables al restarle la media y dividirla por la varianza de la variable.

Con esto logramos homogeneizar las variables y facilitar la comparación entre atributos en el modelo logístico.

Generamos nuevos objetos llamados `X_train_std` y `X_test_std` que guardaran las nuevas matrices. Estos objetos alojarán el resultado de `StandardScaler().fit_transform()`.

```
from sklearn.preprocessing import StandardScaler

# estandarizamos la matriz de entrenamiento
X_train_std = StandardScaler().fit_transform(X_train)
# estandarizamos la matriz de pruebas
X_test_std = StandardScaler().fit_transform(X_test)

# iniciamos el modelo con la clase LogisticRegression y pasamos los
datos en fit.
```

```
default_model = LogisticRegression().fit(X_train_std, y_train)
```

Al solicitar los coeficientes del modelo, observamos que son distintos a los estimados con nuestro modelo econométrico dado que no están estandarizados.

```
default_model.coef_
```

```
array([[-0.33263063,  0.59061437,  0.14254114, -0.05194878]])
```

Procedemos por generar las predicciones en base a nuestro modelo mediante `predict(X_test_std)`. Cabe destacar que los modelos de clasificación presentan fórmulas alternativas de predicción.

Mientras que con la opción `predict` solicitamos las clases en las nuevas observaciones, si solicitamos `predict_proba` obtendremos el mapeo $Pr(y*) \rightarrow [0, 1]$, lo cual nos será útil cuando midamos tasas de falsos positivos y negativos en el modelo. Por último también está la opción `predict_log_proba` que devuelve el logaritmo de la probabilidad generada con `predict_proba`.

```
yhat = default_model.predict(X_test_std)
# solicitemos las primeras 20 observaciones del vector predicho.
yhat[:20]
```

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

Hasta el momento nuestro flujo de trabajo es idéntico al problema de regresión. Resulta que la principal diferencia entre ambas aproximaciones es cómo evaluamos su desempeño.