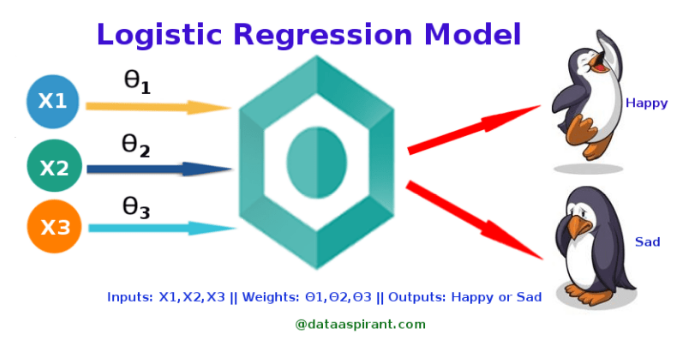
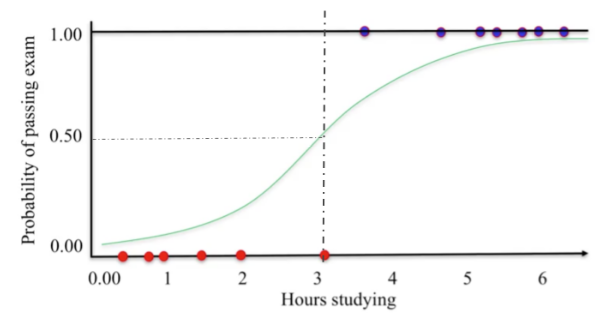
**Regresión Logística**

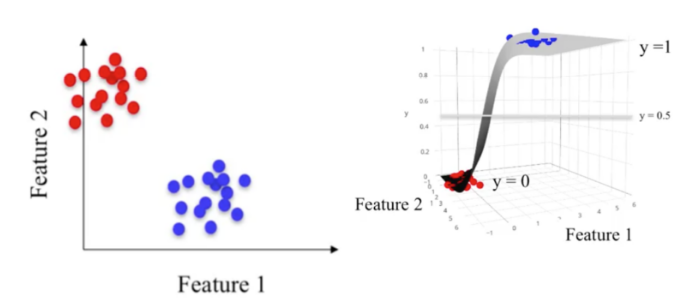


**Introducción:** La **regresión logística** es un **abordaje lineal** para resolver **problemas de clasificación:**

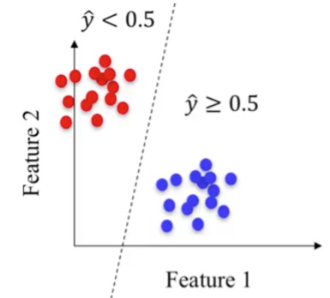
Se podría usar una regresión lineal, pero nos daría valores fuera del rango [0,1] y eso dificultaría la interpretación como probabilidad. La regresión lineal nos permite **modelar la probabilidad** de que la variable objetivo y pertenezca a una determinada categoría, para unos determinados valores de X. Se establece una **frontera de decisión lineal:**



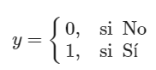
Ejemplo con dos features:



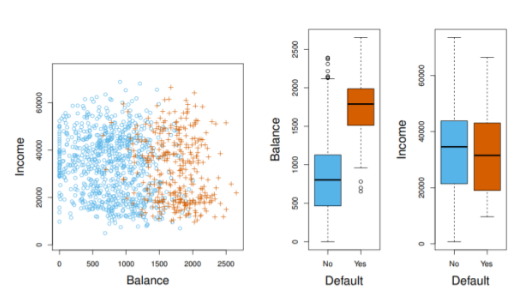
Es posible también definir una frontera que resuelva el problema:



Otro Ejemplo: Queremos **predecir la probabilidad** de que un cliente no pague su Tarjeta de Crédito:



Usando como *Features*, sus ingresos (income) y la deuda de su tarjeta de crédito (balance):



A partir de la observación de los datos, se decide usar como variable predictora el **balance**. Dicho en forma matemática, nos interesa predecir p(y = 1 | balance).

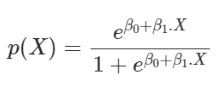
Si definimos como umbral 0.5, si p(y = 1 | balance) > 0.5 Entonces Default.

**Regresión Logística:**

No conviene usar una **regresión lineal**, porque además de **dar valores fuera del rango [0,1]**, en problemas de clasificación multiclase, **el modelo interpretaría las diferentes clases como valores numéricos**.

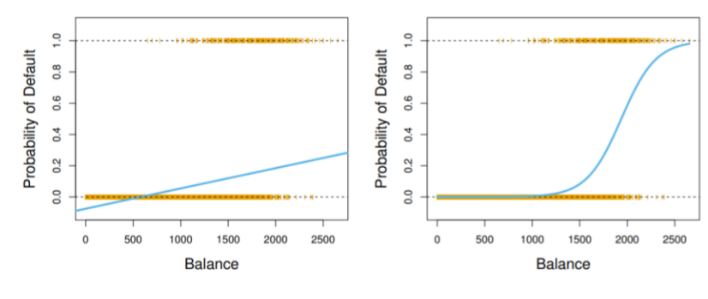
Lo que nosotros **queremos** ahora es **una función que nos garantice** que las **estimaciones** que hagamos estén **dentro del rango válido de una probabilidad**.

Esa función es la **función Logística:**

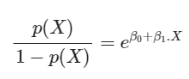


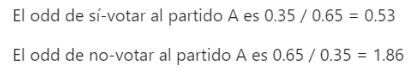
Independientemente de los valores que tome X, siempre va a dar valores dentro del rango [0,1].

Diferencia entre regresión lineal y regresión logística:



**Odd:** Probabilidad de que un evento suceda / probabilidad de que dicho evento no suceda. Oscila entre 0 e infinito. Se pueden calcular tanto para la ocurrencia de un evento como para la no ocurrencia de un evento:

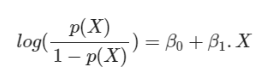




Es 1.86 veces más probable que alguien no sea del partido A de que lo sea.

Es posible **interpretar las odds como ratios**: Cantidad de veces de que algo pueda suceder sobre la cantidad de veces de que pueda no suceder.

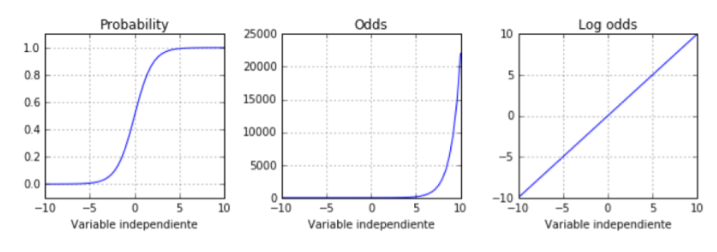
El **logaritmo de odds-ratio** tiene una **relación lineal con X**:



Mientras que en la ***regresión lineal,* β*p es el cambio promedio en y ante un cambio en Xp;*** en la ***regresión logística,* β*1*** *es el* ***cambio del logaritmo del odds-value en y ante un cambio en Xp*.** Dicho en otras palabras, Multiplica el odds por **β*1***.

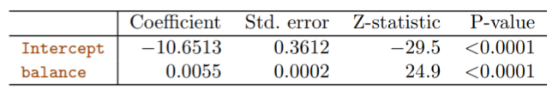
La relación entre p(X) y X no es una línea recta.

El signo de β1 expresa la dirección de cambio de p(X), al margen del valor de X.



Ejemplos:

Volvemos al caso de tarjetas de crédito. Obtenemos estos valores para el modelo ajustado:



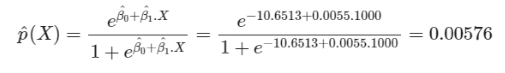
Un incremento de 1$ en el balance incrementa 0.0055 unidades el “Log odds-ratio”. Se usa un estadístico z, análogo al estadístico t en regresión lineal:

H0: β1 = 0 (probabilidad de default no depende del balance)

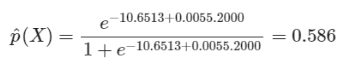
H1: β1 != 0 (probabilidad de default depende del balance)

Como tenemos estimados los coeficientes del modelo, podemos hacer predicciones y computar la probabilidad de default para algún valor dado de balance.

IE:

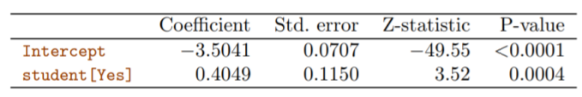
Para un balance de 1000$, tenemos que está por debajo del 1% 

Para un balance de 2000$, es mucho más grande y está cerca del 58%



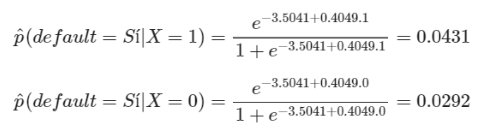
La regresión logística **permite también variables cualitativas**.

Entrenamos un modelo que usa la condición de ser o no estudiante sobre la probabilidad de default:



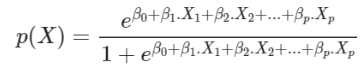
Como el **coeficiente** es **positivo**, esto nos indica que **ser estudiante** tiene una **relación positiva con ser potencial moroso**.

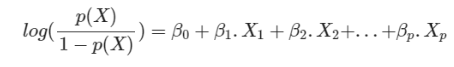
Podemos estimar las probabilidades de entrar en default siendo o no siendo estudiante:



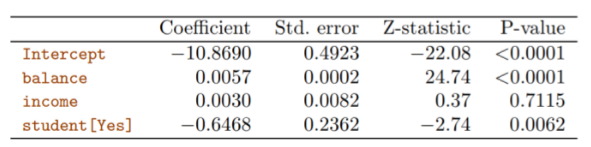
**Regresión Logística Múltiple**

En forma análoga a una regresión lineal múltiple, podemos pensar el problema de predecir una variable cualitativa binaria con una serie de *p* features:

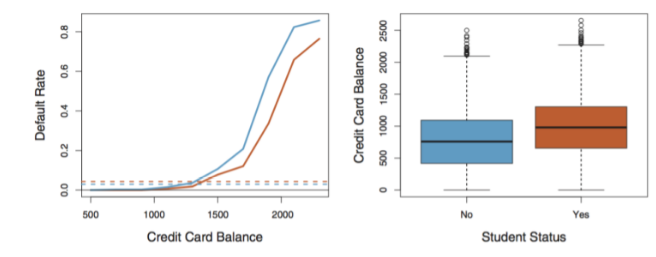




Ejemplo, aplicamos este modelo para predecir la probabilidad de default a partir del ingreso, balance y si es o no estudiante:

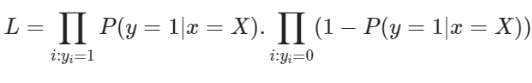


El p value de income es muy alto, lo que nos indica que esta variable no contribuiría significativamente al efecto de defaultear o no.

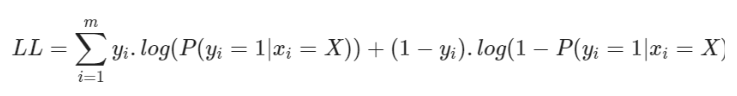


En el cuadro de la izquierda, la línea azul son los no estudiantes, y la línea marrón son los estudiantes. Las líneas punteadas son % total de defaults, las líneas continuas representan el % de defaults en función del balance de la tarjeta de crédito.

**Función de Costo:** Se busca definir una **Función de Costo** que dependa de **los parámetros del modelo.** El **entrenamiento o aprendizaje** consiste en **encontrar los parámetros que la optimicen**. El objetivo en la regresión logística es **maximizar la verosimilitud del modelo;**  que las estimaciones de p(observaciones de clase 1) sean cercanas a 1; y lo mismo para las que pertenezcan a la clase 0.



Tomando el logaritmo:



Cuando yi = 1 se suman los:

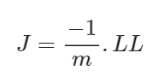
, los cuales deberían resultar en valores cercanos a 1.

Cuando yi = 0 se suman los:

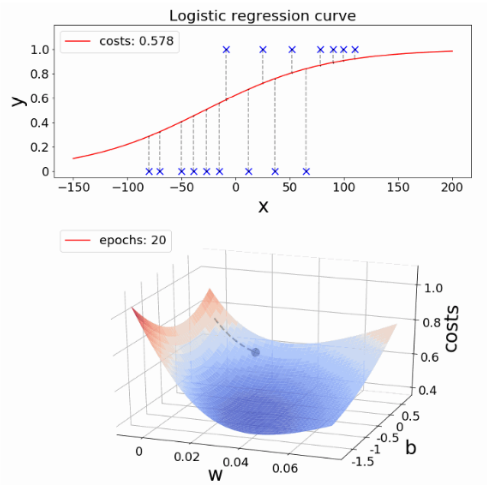
, los cuales deberían resultar en valores cercanos a 0.

Esto se conoce como **Maximum Likelihood Estimation (MLE)**.

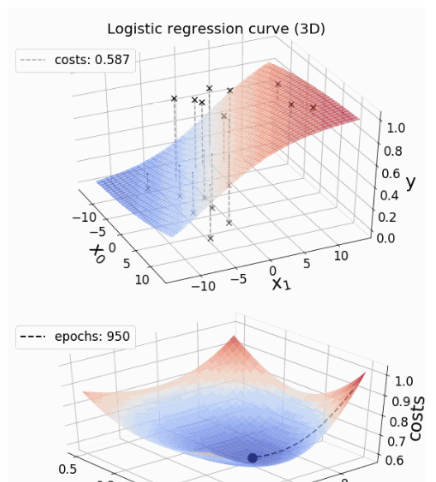
Como la función de costo es algo que busca minimizarse, se trabaja con el opuesto y se promedia:



**Regresión Logística con una Variable Predictora:**

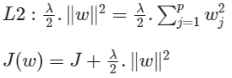


**Regresión Logística con dos Variables Predictoras:**

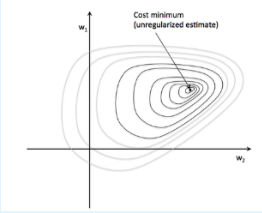


**Regularización:**

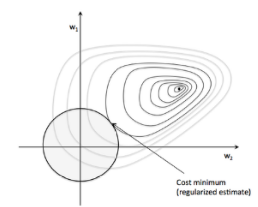
Es posible pensar en **regularización** como **agregar sesgo** si nuestro modelo sufre de **alta varianza** (**overfitting**)**.** Pero un sesgo excesivo dará también un ajuste insuficiente (si tenemos un mal rendimiento tanto para datos de entrenamiento como de testeo, esto puede estar indicándonos que tenemos sesgo excesivo). El objetivo en un modelo no regularizado es minimizar la función de costo: Queremos encontrar los pesos de los features que corresponden al costo mínimo global. Si regularizamos la función de costo, lo que hacemos es agregar un término adicional a la misma, que aumenta a medida que aumenta el valor de los pesos de sus parámetros (w o β). Se agrega un nuevo hiperparámetro λ con la regularización, para controlar la fuerza de la misma:

****

**No regularizado:**

****

**Regularizado:**

****

**Scikit-learn**

Ejemplo: Tenemos un Dataset Default con información sobre diez mil clientes. Vamos a ajustar un modelo de regresión logística al mismo. Queremos predecir qué clientes incumplirán con la deuda de su tarjeta de crédito.

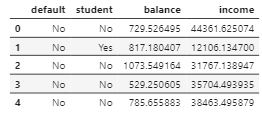
**En Python**, vamos a usar scikit-learn LogisticRegression; que es una clase que implementa la regresión logística regularizada usando la biblioteca “liblinear”, los métodos “newton-cg”, “sag”, “saga”, “lbfgs”. La regularización se aplica por default.

from sklearn.linear\_model import LogisticRegression

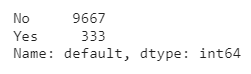
from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, confusion\_matrix

data = pd.read\_csv(‘../Data/Default.csv’, sep=”\t”)

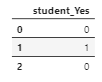


data.default.value\_counts()

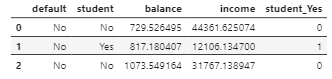


Las categorías están muy desbalanceadas. Vamos a convertir las variables categóricas en Dummies:

Student\_dummy = pd.get\_dummies(data.student, drop\_first = True, prefix = ‘student’)



data = pd.concat([data, student\_dummy], axis = 1)



X = data[[‘student\_yes’, ‘balance’, ‘income’]]

y = data.default

# Se crean los conjuntos de entrenamiento y testeo, estableciendo el valor del parámetro stratify para que los datos se dividan en forma estratificada según las etiquetas de Default. De esta manera, lo que se hace es construir **conjuntos de entrenamiento y testeo con la misma proporción de registros en cada categoría que el DF original**.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, stratify=y, random\_state = 12)

# Lo podemos comprobar:

data.default.value\_counts()[‘Yes’]/ data.shape[0]



y\_train.value\_counts()[‘Yes’]/ y\_train.shape[0]



y\_test.value\_counts()[‘Yes’]/ y\_test.shape[0]



# Ahora instanciamos una regresión logística:

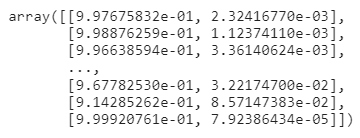
logistic\_regression = LogisticRegression(penalty = ‘none’) # sin regularización

logistic\_regression.fit(X\_train, y\_train)

y\_test\_pred = logistic\_regression.predict(X\_test)



y\_test\_pred\_proba = logistic\_regression.predict\_proba(X\_test)



accuracy\_score(y\_test, y\_test\_pred)



confusion\_matrix(y\_test, y\_test\_pred)



Si bien el accuracy dio alto, la matriz de confusión nos muestra que la performance para la etiqueta ‘Yes’ fue bastante mala: De 83 registros clasificó solamente 17 bien (el 80% de los registros ‘Yes’ dieron mal); pero la performance para la etiqueta ‘No’, en cambio, fue buena: Sólo 13 de 2417 registros fueron mal etiquetados (0.5%).

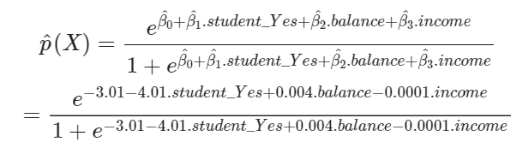
logistic\_regression.intercept\_ # obtenemos β0



logistic\_regression.coef\_



Entonces el modelo termina quedando así:



Tener en cuenta que **no hace falta estandarizar para la regresión logística,** a menos que queramos usar **regresión logística con regularización de Ridge o Lasso** donde **sí es necesario.**