Universidad Nacional de La Plata

Facultad de informática

Matemática 4

Informe Regresión Lineal

Estudiantes:

Castro, Lautaro Germán; 20000/3

Lopez, Gustavo; 19353/1

Índice

Nota: Los fragmentos de código más importantes se encuentran disponibles en este informe. De igual manera si el código desea verse en profundidad, se adjunta un repositorio con todos ellos y el dataset filtrado: Github-Códigos

1. Parte 1: Predicción del valor de mercado

a) Recta de regresión para predecir el valor de mercado de un jugador a partir de la característica más relevante (a la que se destinará mayor proporción del presupuesto), respaldada por:

1.1. (i)

Prueba de significancia de regresión, coeficiente de determinación (R^2) y correlación lineal (r).

Para poder obtener una recta de regresión lineal que prediga el valor de mercado de un jugador, primero vamos a conseguir el atributo que mejor prediga el valor de este.

Lo primero que hacemos es limpiar las columnas que no nos sirvan para hacer el análisis y quedarnos solo con algunas que resumen las características del jugador. Decidimos eliminar las columnas de puntaje de habilidades particulares ya que estas tienen muchos valores vacíos, lo cual dificulta el análisis y por otro lado se pueden resumir en las columnas de overall y potential.

```
file_path = "players_21.csv"
df = pd.read_csv(file_path)

columns_to_keep = [
    "age",
    "height_cm",
    "weight_kg",
    "overall",
    "potential",
    "value_eur",
    "wage_eur",
    "skill_moves",
]
df_filtered = df[columns_to_keep]
```

Código 1: Filtrado de columnas de interés

Una vez hecho el filtrado, realizamos un análisis sobre cada atributo y vemos cuál nos da una mayor fuerza de correlación con el valor del jugador (y), para luego poder realizar un análisis sobre este.

Para eso primero necesitamos una función que nos calcule el coeficiente r, para esto directamente hicimos una función que calcula en general todos los coeficientes y estimadores para que sea lo más reutilizable posible.

```python
def calculate_stats(x, y):
    x = np.array(x)
    y = np.array(y)

    n = x.size
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    sumxy = np.sum(x * y)
    sumx = np.sum(x)
    sumx2 = np.sum(x**2)
    sumy = np.sum(y)

    Sxy = sumxy - (sumx * sumy) / n
    Sxx = sumx2 - sumx**2 / n
    Syy = np.sum((y - y_mean) ** 2)

    b1 = Sxy / Sxx
    b0 = y_mean - b1 * x_mean

    regy = b0 + b1 * x

    SCE = np.sum((y - regy) ** 2)
    SCR = np.sum((regy - y_mean) ** 2)
    STC = Syy

    R2 = 1 - (SCE / STC)  # o también SCR / STC

    alpha = 0.05
    t = t_student.ppf(1 - alpha / 2, n - 2)

    r = Sxy / np.sqrt(Sxx * Syy)
    sigma2 = SCE / (n - 2)

    T = b1 / np.sqrt(sigma2 / Sxx)

    return {
        "n": n, "x_mean": x_mean, "y_mean": y_mean, "Sxy": Sxy,
        "Sxx": Sxx, "Syy": Syy, "b0": b0, "b1": b1,
        "SCE": SCE, "SCR": SCR, "R2": R2, "r": r,
        "sigma2": sigma2, "T": T, "t": t,
    }
```

Código 2: Cálculo general de coeficientes y estimadores

Luego utilizando esta función vamos a iterar por cada columna filtrada, y nos quedamos con el que mayor r nos de.

```
r_values = {}
stats = {}
y_column = "value_eur"
y_values = df_filtered[y_column].values

for x_column in columns_to_keep:
    if x_column != y_column:
        x_values = df_filtered[x_column].values
        stats = calculate_stats(x_values, y_values)
        r_values[x_column] = stats["r"]

best_x_column = max(r_values, key=r_values.get)
best_r_value = r_values[best_x_column]

for key, value in sorted(r_values.items(), key=lambda item: item[1], reverse=True):
    print(f"Columna: {key} -> r: {value}")

print(f"La columna '{best_x_column}' produce el mayor valor de r: {best_r_value}")
```

Código 3: Calcula el x con mayor r

Nos queda los siguientes datos:

- wage_eur: 0.840126
- overall: 0.630085
- potential: 0.571925
- skill_moves: 0.293722
- age: 0.084973
- weight_kg: 0.044709
- height_cm: 0.004123

La columna 'wage_eur' produce el mayor valor de r: 0.840126

Como se puede observar, el sueldo del jugador tiene una fuerte incidencia sobre el valor del jugador. Ahora sabiendo con certeza que atributo es el que mayor correlación tiene, lo respaldamos con una prueba de significancia de regresión, haciendo un test de hipótesis sobre $\hat{\beta}_1$, sobre el caso especial $H_\theta : \beta 1 = 0$ (Esto se calcula en Código 2) y lo evaluamos:

```python
def significance_test(data):
    T = np.abs(data["T"])
    t = data["t"]

    if T > t:
        results = "x tiene cierta influencia sobre y"
    else:
        results = "x no tiene influencia sobre y"

    return results

significance_res = significance_test(stats)
print("Test de significancia: ", significance_res)
```

Código 4: Test de significancia

Resultado: Test de significancia: x tiene cierta influencia sobre y

Como se rechaza la hipótesis de h0 entonces sabemos que la pendiente de la recta de regresión verdadera no es nula o cero y hay una cierta influencia. Luego obtenemos también los coeficientes de correlación y determinación (calculados en Código 2).

```python
stats = calculate_stats(df_filtered[best_x_column].values, y_values)
print(
    "Coeficientes de correlación lineal r y de determinación r² son: ",
    stats["r"],
    stats["R2"],
)
```

Código 5: Coeficientes de regresión y determinación

Finalmente, los coeficientes de correlación lineal r y de determinación R^2 son: 0.8401260510834834, 0.7058117817091276

Tomando los estimadores $\hat{\beta}_0$ y $\hat{\beta}_1$ que quedaron guardados en 'stats' la recta de regresión queda:

$$\hat{y} = 332598{,}821628 + 218{,}101272x$$

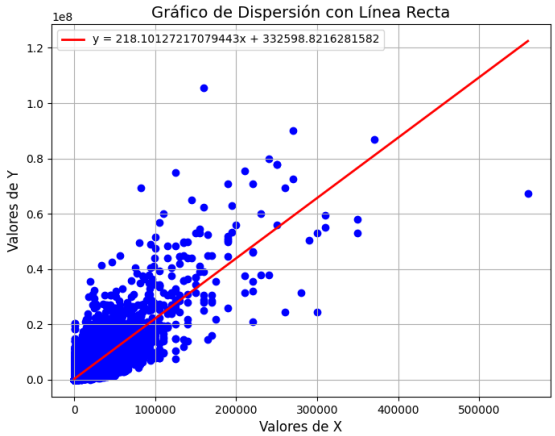

Figura 1.1: Modelo estimado con regresión lineal simple

1.2. (ii)

ii) Inferencias sobre los parámetros de la recta, estimando las fluctuaciones con una confianza del 95 %.

En base a lo anterior, podemos estimar las fluctuaciones de los parámetros b0 y b1 utilizando los intervalos de confianza definidos en la teoría para β_0 y β_1.

```python
def estimateFluctiations(data):
    b0 = data["b0"]
    b1 = data["b1"]
    t = data["t"]
    sigma2 = data["sigma2"]
    Sxx = data["Sxx"]
    n = data["n"]
    x_mean = data["x_mean"]

    variance_b1 = sigma2 / Sxx
    variance_b0 = sigma2 * (1 / n + (x_mean**2) / Sxx)

    ini_conf_b1 = b1 - t * np.sqrt(variance_b1)
    end_conf_b1 = b1 + t * np.sqrt(variance_b1)

    ini_conf_b0 = b0 - t * np.sqrt(variance_b0)
    end_conf_b0 = b0 + t * np.sqrt(variance_b0)

    print(f"Intervalo de confianza para b1: ({ini_conf_b1}, {end_conf_b1})")
    print(f"Intervalo de confianza para b0: ({ini_conf_b0}, {end_conf_b0})")

estimateFluctiations(stats)
```

Código 6: Inferencias sobre el parámetro de la recta

Intervalo de confianza para b1: (216.095926, 220.106618)
Intervalo de confianza para b0: (289516.064534, 375681.578721)

1.3. (iii)

iii) La proporción de veces que el valor de mercado supera la incertidumbre de predicción comparada con la respuesta media del valor de mercado para una característica fija, ambas con la misma confianza y ancho mínimo.

En primer lugar, para llevar a cabo el análisis, se utilizan dos conceptos clave:

- **Intervalo de confianza para la respuesta media:** Este intervalo establece un rango para la media de una predicción cuando se fija un valor denominado x^*. Así, la fórmula correspondiente es:

$$IC(x^*) = \left[\hat{\beta}_0 + \hat{\beta}_1 x^* - t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}\right)}, \quad \hat{\beta}_0 + \hat{\beta}_1 x^* + t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(\frac{1}{n} + \frac{(x^* - \bar{x})^2}{S_{xx}}\right)}\right]$$

  De esta manera a través de este intervalo se puede establecer un rango para las posibles respuestas medias cuando se fija un valor.

  - Una observación importante es que El ancho del intervalo será mínimo cuando $x^* = \bar{x}$.

- **Intervalo de predicción para valores futuros de Y:** Este intervalo establece un rango para valores factibles de una predicción. Así, la fórmula correspondiente es:

$$IP(x^*) = \left[\hat{Y}^* - t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(1+\frac{1}{n}+\frac{(x^*-\bar{x})^2}{S_{xx}}\right)};\quad \hat{Y}^* + t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(1+\frac{1}{n}+\frac{(x^*-\bar{x})^2}{S_{xx}}\right)}\right]$$

  Así, este intervalo permite establecer un rango de incertidumbre ya que define los posibles valores para una predicción.

Una vez definidos los conceptos clave a utilizar, se establecen los parámetros para la construcción de los intervalos:

$$\alpha = 0{,}05 \qquad x^* = \bar{x}$$

De modo que para calcular la proporción puede realizarse de dos maneras: **midiendo las proporciones algebraicamente, una solución a través de software**

Solución algebraica

Sean:

$$\epsilon IC(x^*) = t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(\frac{1}{n}+\frac{(x^*-\bar{x})^2}{S_{xx}}\right)}$$

$$\epsilon IP(x^*) = t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(1+\frac{1}{n}+\frac{(x^*-\bar{x})^2}{S_{xx}}\right)}$$

$$\begin{aligned}\frac{\epsilon IP(\bar{x})}{\epsilon IC(\bar{x})} &= \frac{t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(1+\frac{1}{n}+\frac{(\bar{x}-\bar{x})^2}{S_{xx}}\right)}}{t_{\frac{\alpha}{2},n-2}\sqrt{\sigma^2\left(\frac{1}{n}+\frac{(\bar{x}-\bar{x})^2}{S_{xx}}\right)}} = \frac{\sqrt{\sigma^2\left(1+\frac{1}{n}+\frac{(\bar{x}-\bar{x})^2}{S_{xx}}\right)}}{\sqrt{\sigma^2\left(\frac{1}{n}+\frac{(\bar{x}-\bar{x})^2}{S_{xx}}\right)}} \\ &= \frac{\sqrt{\sigma^2\left(1+\frac{1}{n}+0\right)}}{\sqrt{\sigma^2\left(\frac{1}{n}+0\right)}} = \sqrt{\frac{\sigma^2\left(1+\frac{1}{n}\right)}{\sigma^2\left(\frac{1}{n}\right)}} \\ &= \sqrt{\frac{(1+\frac{1}{n})}{1} \div \frac{1}{n}} = \sqrt{\frac{n(1+\frac{1}{n})}{1}} \\ &= \sqrt{n+1}\end{aligned}$$

Solución por software

Para automatizar el proceso de cálculo, simplemente deben aplicarse la siguiente lógica:

1) Calcular los estadísticos correspondientes

2) Armar los intervalos en x^*, es decir $IC(x^*)$ e $IP(x^*)$

3) Calcular las longitudes $L = lim.superior - lim.inferior$

4) Calcular la proporción: L_{IP}/L_{IC}

Los detalles de implementación pueden verse en Código 7.

```python
def calcule_ic(n, xstar, alpha, b0, b1, x_mean, Sxx, sigma2):
    t = calcule_t(n, 2, alpha / 2)
    y_hat = b0 + b1 * xstar
    wide = t * np.sqrt(sigma2 * (1 / n + (xstar - x_mean) ** 2 / Sxx))
    ic = [y_hat - wide, y_hat + wide]
    return ic

def calcule_ip(n, xstar, alpha, b0, b1, x_mean, Sxx, sigma2):
    t = calcule_t(n, 2, alpha / 2)
    y_hat = b0 + b1 * xstar
    wide = t * np.sqrt(sigma2 * (1 + 1 / n + (xstar - x_mean) ** 2 / Sxx))
    ip = [y_hat - wide, y_hat + wide]
    return ip

s = calculate_stats(x, y) # Código implementado en el inciso (i)
ip = calcule_ip(
    n=s["n"], xstar=s["x_mean"],  alpha=0.05,
    b0=s["b0"],  b1=s["b1"],
    x_mean=s["x_mean"],  Sxx=s["Sxx"],  sigma2=s["sigma2"],
)
ic = calcule_ic(
    n=s["n"],  xstar=s["x_mean"], alpha=0.05,
    b0=s["b0"],  b1=s["b1"],  x_mean=s["x_mean"],
    Sxx=s["Sxx"], sigma2=s["sigma2"],
)
L_ic = ic[1] - ic[0]
L_ip = ip[1] - ip[0]
print(f"La proporción es: {L_ip / L_ic}\n")
```

Código 7: Cálculo de intervalos y proporción

Conclusiones

Finalmente los resultados fueron los siguientes:

- Solución por software: 137,64083696345315
- Solución algebraica: $\sqrt{18944 + 1} = 137{,}640837$

Así, estos resultados indicarían que existe un grado de incertidumbre muy alto en la predicción, debido a que el intervalo de predicción es aproximadamente 137 veces más amplio que el intervalo de confianza. Esto indicaría que la predicción del valor de mercado individual presenta una alta variabilidad, implicando que predicciones basadas en la media podrían subestimar el verdadero valor de mercado, debido a esta amplia diferencia en los intervalos.

2. Parte 2: Ecuación de predicción del valor de mercado

2.1. (i)

Usando el método de mínimos cuadrados. Explica los indicadores obtenidos (como el coeficiente de determinación y la correlación) y proporciona una breve interpretación de los resultados.

En concordancia con lo explicado en el ejercicio 1) se tomaron esos mismos atributos para el análisis, en este caso para realizar un modelo con regresión múltiple.

La lógica para una solución por software es la siguiente:

1) Generar las matrices X e Y con las observaciones correspondientes del dataset.

2) Calcular la estimación es decir, $\hat{\beta} = \left(X^T X\right)^{-1} X^T Y$

3) Calcular los indicadores para el modelo de correlación, es decir R^2 y r

Para la estimación de los coeficientes utilizamos esa ecuación debido a que nuestra matriz no es cuadrada, y sería la forma más conveniente para calcular el estimador.

Los detalles de implementación pueden verse en Código 8 (Cálculo de coeficientes) y Código 9 (Cálculo de indicadores).

Los datos fueron previamente normalizados debido a que la escala de estos es grande y esto hace al algoritmo propenso a errores en los rangos del tipo de número y en la gestión de memoria.

```
# Carga de las matrices X e Y con el dataset ...

# Agregar columna de unos para el término independiente
X = np.concatenate([np.ones((X.shape[0], 1)), X], axis=1)

XTX = np.dot(X.T, X)
XTX_inv = np.linalg.inv(XTX)
XTy = np.dot(X.T, y)
beta = np.dot(XTX_inv, XTy)
```

Código 8: Cálculo de coeficientes en regresión múltiple

```
y_hat = np.dot(X, beta)
n = df.shape[0] # se obtiene la cantidad de observaciones
k = beta.shape[0] # Se obtiene la cantidad de varibles independientes
SSr = np.sum((y - y_hat) ** 2)
STC = np.sum((y - np.mean(y)) ** 2)
R2 = 1 - SSr / STC
R2a = 1 - (1 - R2) * ((n - k) / (n - k - 1))
r = np.sqrt(R2a)
```

Código 9: Cálculo de indicadores para modelo de regresión múltiple

Conclusiones e interpretación

En primer lugar, los resultados de los análisis fueron los siguientes:

- $\hat{\beta}_0 = 2224813{,}291807432$ (constante)
- $\hat{\beta}_1 = -866214{,}5668436177$ (age)
- $\hat{\beta}_2 = -51508{,}2841879795$ (height_cm)
- $\hat{\beta}_3 = 24826{,}29246999719$ (weight_kg)
- $\hat{\beta}_4 = 1605370{,}8391333632$ (overall)
- $\hat{\beta}_5 = 11007{,}493776507676$ (potential)
- $\hat{\beta}_6 = 3496653{,}2299954006$ (wage_eur)

- $\hat{\beta}_7 = -4685{,}965387084987$ (skill_moves)
- $r = 0{,}8721742466048612 \qquad R_a^2 = 0{,}7606879164407572$

De esta manera, a través de los coeficientes (los betas) se reflejan muchos razonamientos intuitivos, como por ejemplo con overall, potential donde claramente el potencial o calificación general afectan positivamente (pendiente positiva) el valor de mercado, o aún más wage_eur (sueldo mensual) que claramente está estrechamente relacionado con el valor de mercado, luego características como la edad (age) son coherentes ya que a medida que un jugador envejece su valor va decayendo en general, luego quizá height(altura) y peso (weight) no presentan razonamientos tan intuitivos, pero aún así el modelo nos permite evidenciar que la altura influye negativamente y el peso positivamente.

Finalmente los indicadores expresan que el modelo generado es correcto y adecuado, así el coeficiente de correlación r = 0.872 denota una fuerte correlación positiva y luego el coeficiente de determinación ajustado $R_a^2 = 0{,}760$ explica la variabilidad del valor de mercado en un 76 % respecto al modelo, lo cual es bastante bueno. De esta manera, se refleja lo que se intuye naturalmente, el valor de mercado no es algo que se determine de forma absoluta por un único factor, sino que está definido por múltiples factores, por ello el modelo de regresión simple no resultó ser tan acertado a diferencia del modelo de regresión múltiple.

2.2. (ii)

Usando el método de descenso por gradiente. ¿Son los valores obtenidos iguales a los conseguidos mediante la resolución del sistema de ecuaciones normales? Muestra los resultados obtenidos junto con las últimas iteraciones del algoritmo. Indica los valores de los parámetros utilizados (como tasa de aprendizaje y número de iteraciones).

Para generar un modelo de regresión múltiple por descenso de gradiente, se plantea el error cuadrático medio (ECM) para minimizar la función:

$$ECM = \frac{1}{m}\sum_{i}^{m}(y_i - \hat{y}_i)^2 = \frac{1}{m}SCE$$

Como puede observarse ECM no es más que el error cuadrático o suma de los cuadrados de los residuos dividido por m que es la cantidad de observaciones de la muestra, y se utiliza en el método de descenso de gradiente sobre el SCE con el objetivo de que los errores sean proporcionales a la cantidad de observaciones ya que de no hacerlo los valores crecerían de acuerdo al tamaño de la muestra. En este caso se utiliza una variante que en vez de dividir por $\frac{1}{m}$ se divide por $1/2m$, lo cual permite simplificar el cálculo del vector gradiente que finalmente en su versión matricial es:

$$\frac{1}{2m}\sum_{i}^{m}(\hat{y}_i - y_i)^2 \;\Rightarrow\; \nabla f(B) = \frac{1}{m}X^T(\hat{y} - Y)$$

Así, finalmente puede implementarse el siguiente programa:

```
def ecm(b):
    y_est = X.dot(b)
    error = y_est - Y
    return (1 / (2 * n)) * np.sum(error**2)

def ecm_grad(b):
    y_est = X.dot(b)
    error = y_est - Y
    return (1 / n) * X.T.dot(error)
```

Código 10: Función y vector gradiente para descenso de gradiente

```
f = ecm
df = ecm_grad
tol = 1e-6
step = 0.01
iteration = 1

beta = np.zeros(X.shape[1])  # punto inicial x_0 (0, 0, ..., 0)
f_prev = f(beta)
beta_new = beta - step * df(beta)
f_new = f(beta_new)
error = abs(f_new - f_prev)

while error > tol:
    beta, f_prev = beta_new, f_new

    beta_new = beta - step * df(beta)
    f_new = f(beta_new)
    error = abs(f_new - f_prev)

    iteration += 1
```

Código 11: Algoritmo de descenso de gradiente

Resultados

- $\varepsilon =$ 1e-6 $\quad \eta = 0.01 \quad x_0 = (0, 0, 0, ..., 0)$
- $\hat{\beta}_0 = 2224813{,}2918074094$
- $\hat{\beta}_1 = -866211{,}9643719553$
- $\hat{\beta}_2 = -51508{,}35019288312$
- $\hat{\beta}_3 = 24826{,}431573733447$
- $\hat{\beta}_4 = 1605367{,}2119736625$
- $\hat{\beta}_5 = 11010{,}530098933274$
- $\hat{\beta}_6 = 3496653{,}3808589084$
- $\hat{\beta}_7 = -4685{,}647051198054$

Como puede observarse las aproximaciones son muy cercanas a las calculadas con el método de mínimos cuadrados.

Iteración 18046

- $\text{f}(x_i) = 3114959565313.2725 \quad \text{f}(x_{i+1}) = 3114959565313.272$
- error $= 0.00048828125$

Iteración 18047

- $\text{f}(x_i) = 3114959565313.272 \quad \text{f}(x_{i+1}) = 3114959565313.27$
- error $= 0.001953125$

Iteración 18048

- $\text{f}(x_i) = 3114959565313.27 \quad \text{f}(x_{i+1}) = 3114959565313.2686$

- error = 0.00146484375

Iteración 18049

- f(x_i) = 3114959565313.2686 f(x_{i+1}) = 3114959565313.266
- error = 0.00244140625

Iteración 18050

- f(x_i) = 3114959565313.266 f(x_{i+1}) = 3114959565313.266
- error = 0.0, el resultado es un redondeo del lenguaje de un número muy cercano a cero que supera

2.3. (iii)

Da una interpretación del criterio de corte utilizado en el algoritmo del gradiente. Explica si presenta alguna falla. Si no es una buena condición de corte, ¿puedes sugerir un criterio alternativo más eficaz?

El criterio de corte que se utiliza (Código 11) está basado en la diferencia entre dos evaluaciones sucesivas en la función de costo. Cuando esta diferencia es menor que un valor de tolerancia previamente establecido, se considera que se alcanzó el mínimo. Este criterio presenta algunos inconvenientes:

- Si la escala de datos es pequeña, puede llegar a detener el algoritmo antes de tiempo, lo cual se puede solucionar dependiendo el dataset con ir ajustando la tolerancia en base a los datos
- Por otro lado no se tiene en cuenta el valor del gradiente, puede llegar a cortar el algoritmo antes que llegue a un valor cercano a cero o también que llegue a un valor cercano a cero pero siga iterando hasta conseguir un valor de tolerancia adecuado, lo cual haría que termine haciendo una cantidad mayor de iteraciones.

Un criterio más eficaz puede ser uno basado en la magnitud del gradiente: Tomando en cuenta el valor del gradiente, podemos analizar cuanto nos falta para llegar a un mínimo, estableciendo también un umbral de tolerancia predefinido para asegurarse que llegue a un valor cercano a cero. También se podría utilizar un método que una ambos criterios descritos anteriormente para analizar tanto la diferencia de iteraciones sucesivas en la función de costo y además asegurarse que cuando se llegue a un valor de gradiente cercano a cero corte el algoritmo sin realizar iteraciones extra. De esta forma se puede hacer más eficiente el algoritmo evitando así que haga computo adicional o que corte antes de tiempo.

3. Parte 3: Comportamiento del método de descenso por gradiente

3.1. (c)

Convergencia del método de descenso por gradiente. Explicar si el método siempre converge al mínimo de la función. En caso contrario, proporciona un contraejemplo para ilustrar este comportamiento.

El método de descenso por gradiente no siempre converge al mínimo de la función, vamos a ver que factores influyen en esto:

1. Función no convexa: Si la función es convexa, el método garantiza la convergencia a un mínimo global debido a que estas funciones tienen un solo mínimo, por ende si la función no es convexa, no se puede garantizar que el algoritmo encuentre un mínimo local en lugar de un mínimo global al detenerse.
2. Tamaño de paso elevado: Si el tamaño de paso es demasiado grande puede producir que oscilen demasiado los valores producidos y el algoritmo puede llegar a quedar trabado y no converger debido a que se excede al mínimo en cada ocasión.

3. Tamaño de paso pequeño: Si el tamaño de paso es demasiado pequeño puede darse que nunca llegue a converger, debido a que las actualizaciones que haga el algoritmo van a ser insignificantes y el tiempo dado puede tender a infinito.

4. Función no diferenciable: Si la función no es diferenciable, el algoritmo falla debido a que el gradiente puede no estar definido y no se pueda utilizar la formula de actualización.

Contraejemplo: Considerando la siguiente función

$$f(x) = x^2$$

Si el tamaño de paso es muy grande, el descenso por gradiente puede oscilar tanto que provoca que el valor de x cada vez se aleje cada vez más del mínimo.

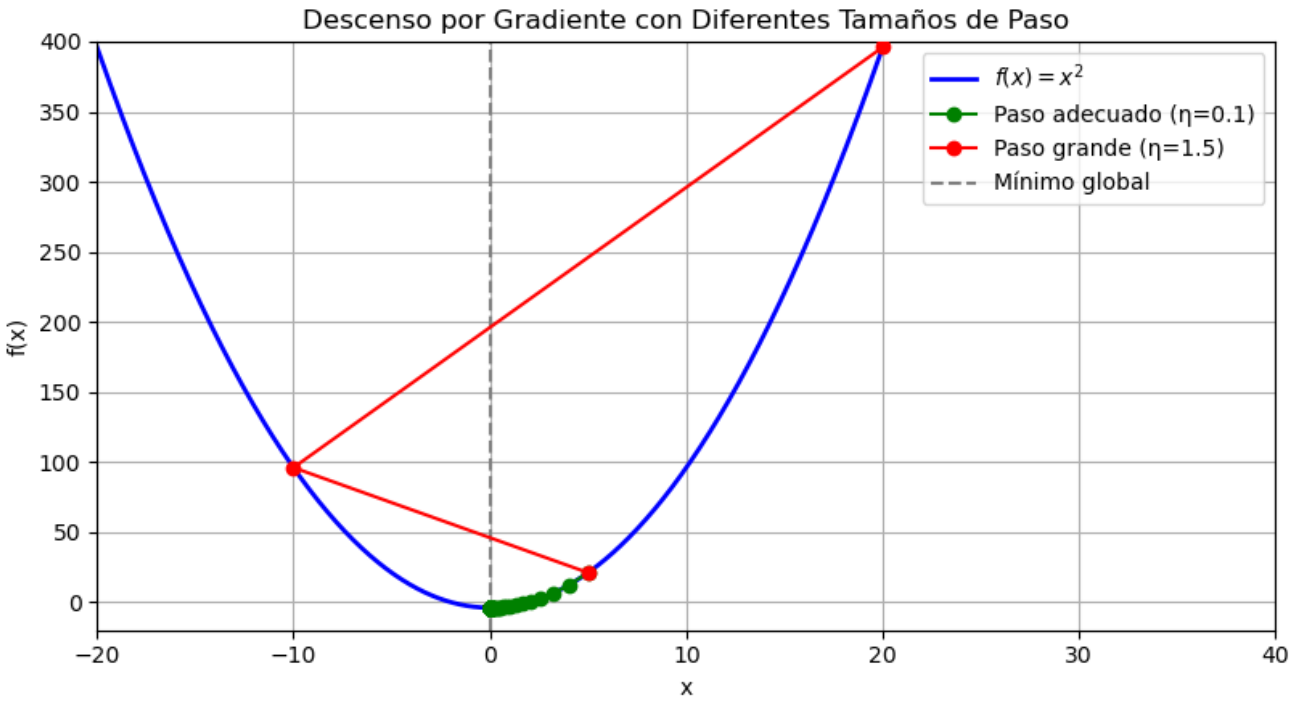

Figura 3.1: Contraejemplo con paso excesivo

Como se puede observar el algoritmo con un paso grande inicia en un punto y luego desde la segunda iteración se aleja cada vez más del mínimo global, haciendo que nunca converja. Por otro lado podemos ver como el mismo algoritmo con un tamaño de paso adecuado converge al mínimo correctamente