Práctica de la sesión 2: evaluación de modelos

Minería de datos II-Universidad Francisco de Vitoria

Pablo Hidalgo (pablo.hidalgo@ufv.es)

En esta práctica vamos a utilizar el modelo de k vecinos más cercanos (en inglés, k-nearest neighbors o KNN) para entender cómo afecta la configuración del modelo al error y, en particular, su impacto en el **underfitting** y **overfitting**.

En el algoritmo KNN el único **hiperparámetro** es el número k de vecinos que utilizamos para realizar una predicción.

> **Antes de empezar la práctica intenta responder a estas dos preguntas:**
>
> - **¿Cómo sería la predicción de un k-vecinos con $k = 1$?**
> - **¿Cómo sería la predicción de un k-vecinos con $k = n$? *(Recuerda que con n nos referimos al número de observaciones en entrenamiento).***

1 ¿Qué necesitas?

Para hacer esta práctica necesitas bajarte de Canvas los archivos:

- `enunciado.pdf`: es un pdf con el desarrollo de la práctica *(¡si estás leyendo esto es que ya lo tienes descargado!)*.
- `train.csv` y `test.csv`: contienen los datos para entrenar y testear los modelos, respectivamente.
- `aux.R`: es un script con funciones que utilizaremos en la práctica.

2 Carga de dependencias y datos

Como en casi cualquier script de R necesitaremos cargar algunos paquetes. En este caso necesitaremos los paquetes `ggplot2`para realizar los gráficos y `FNN` para entrenar un modelo KNN.

```
# Paquete para gráficos
library(ggplot2)

# Paquete para modelo de KNN
library(FNN)
```

Además, tenemos un script auxiliar `aux.R` que contiene algunas funciones ya creadas para que sea más fácil alcanzar el objetivo de esta práctica. Para cargar en la memoria de R funciones que están en un archivo externo, debemos utilizar la función `source()` indicando cuál es la ruta en la que se encuentra el script.

```
# Funciones para facilitar el entrenamiento
source('02-evaluacion_modelos/practica/aux.R')
```

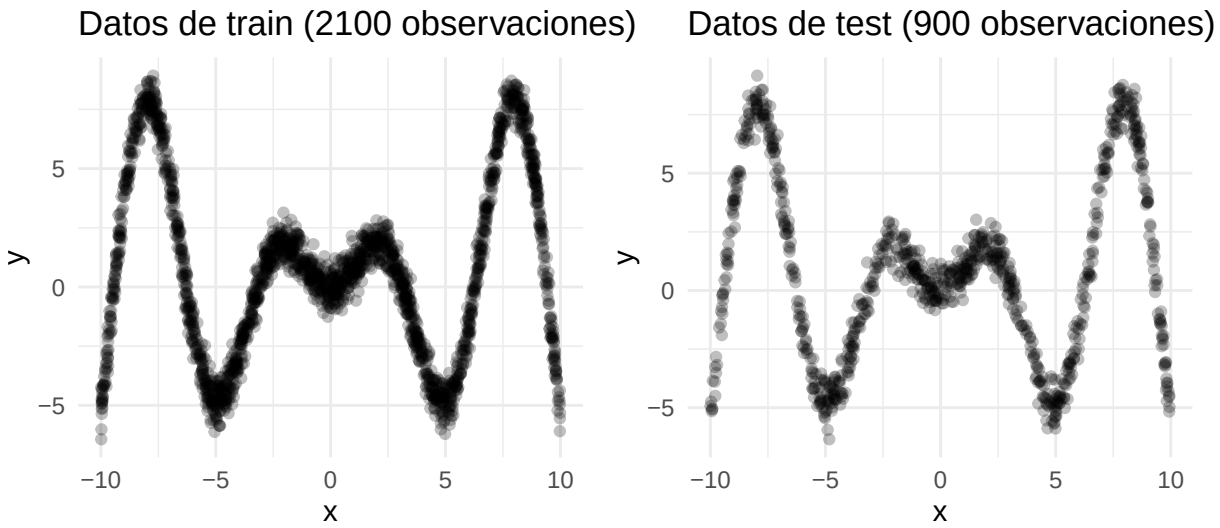
Por último, debemos cargar los datos con los que vamos a trabajar

```
train <- read.csv("02-evaluacion_modelos/practica/data/train.csv")
test <- read.csv("02-evaluacion_modelos/practica/data/test.csv")
```

Ambos conjuntos de datos (recuerda que, en R, nos referimos a ellos como *data frames*) tienen dos variables: la variable predictora `x` y la variable objetivo `y`. Como estos conjuntos de datos tienen solamente dos dimensiones, podemos representarlos gráficamente mediante un diagrama de dispersión.

> Recuerda que, por convenio, siempre representamos en el **eje x (horizontal) la variable predictora** y en el **eje y (vertical) la variable objetivo**.

```
ggplot(train, aes(x = x, y = y)) +
  geom_point(alpha = .25) +
  labs(title = paste0('Datos de train (', nrow(train), ' observaciones)')) +
  theme_minimal()

ggplot(test, aes(x = x, y = y)) +
  geom_point(alpha = .25) +
  labs(title = paste0('Datos de test (', nrow(test), ' observaciones)')) +
  theme_minimal()
```



En la representación gráfica podemos ver varias cosas:

1. Tanto la variable X como la variable Y son cuantitativas. Por lo tanto, se trata de un **problema de regresión.**
2. La forma del patrón que sigue los datos es **altamente no lineal**. Esto significa que un modelo como la regresión lineal sería una mala idea.
3. El patrón que siguen los datos de entrenamiento y de test el similar. Además, este patrón es bastante claro y el *nivel de ruido es bajo*.

4. Si juntásemos ambos conjuntos de datos en uno, el 70% de los datos serían de entrenamiento y el 30% de test.

En este caso como solo tenemos una variable predictora y una variable objetivo, podemos representar gráficamente los datos; en general esto no será posible. Habitualmente tendremos un número de variables predictoras que hará imposible representar las observaciones fácilmente y no es fácil ser conscientes de cómo son los datos.

3 KNN

La única configuración que permite el modelo de k vecinos más cercanos es el número k de vecinos. En esta práctica vamos a ver el efecto que tiene este parámetro (o hiperparámetro) sobre la predicción en test. Como ya sabemos, al tener un conjunto de datos de dos dimensiones, podemos visualizar el efecto fácilmente.

El paquete que utilizaremos es **FNN** y su función `knn.reg()`. Si accedes a la ayuda (puedes escribir `?knn.reg` en la consola) verás que hay que introducir en la función por separado las variables predictoras y la variable objetivo.

```
# Preparación de datos para knn.reg()
train_x <- train['x']
train_y <- train$y

test_x <- test['x']
test_y <- test$y
```

Comencemos con un valor de $k = 500$. Recuerda que el modelo de k vecinos se trata de un modelo *online*. Cuando entrenábamos, por ejemplo, una regresión lineal, primero construíamos el modelo (calculábamos los parámetros) con la función `lm()` y después utilizábamos la función `predict()` para calcular la predicción (al final, una regresión lineal no son más que sumas y multiplicaciones). En este sentido, la regresión lineal se trataba de un modelo *offline* porque el modelo se puede tener previamente calculado *offline* y las predicciones se pueden hacer en cualquier momento; dicho de otra forma, una vez que tenemos los parámetros de la regresión podemos desechar el conjunto de train ya que estos parámetros definen completamente al modelo. Sin embargo, en un KNN **no existe ese concepto de entrenamiento previo**: cada vez que queramos hacer una predicción de una observación debemos calcular los k vecinos más cercanos utilizando el conjunto de entrenamiento completo. Por eso lo denominamos *online*.

Esto significa que el propio modelo ya nos devuelve la predicción. Vamos a calcular la predicción que daría el modelo con $k = 500$ para el conjunto de entrenamiento y el conjunto de test. Calcular los errores cometidos en ambas predicciones nos ayuda a comprobar si el modelo tiene problemas de *overfitting*.

```
# knn.reg() devuelve una lista.
# El elemento $pred de la lista contiene la predicción para el conjunto de test.
pred_train <- knn.reg(train = train_x, test = train_x, y = train_y, k = 500)$pred

pred_test <- knn.reg(train = train_x, test = test_x, y = train_y, k = 500)$pred
```

En el script `aux.R` hay una función denominada `rmse()` que nos permite calcular el RMSE. Vamos a utilizarla para calcular el RMSE en train y test.

```
rmse(train_y, pred_train)
```

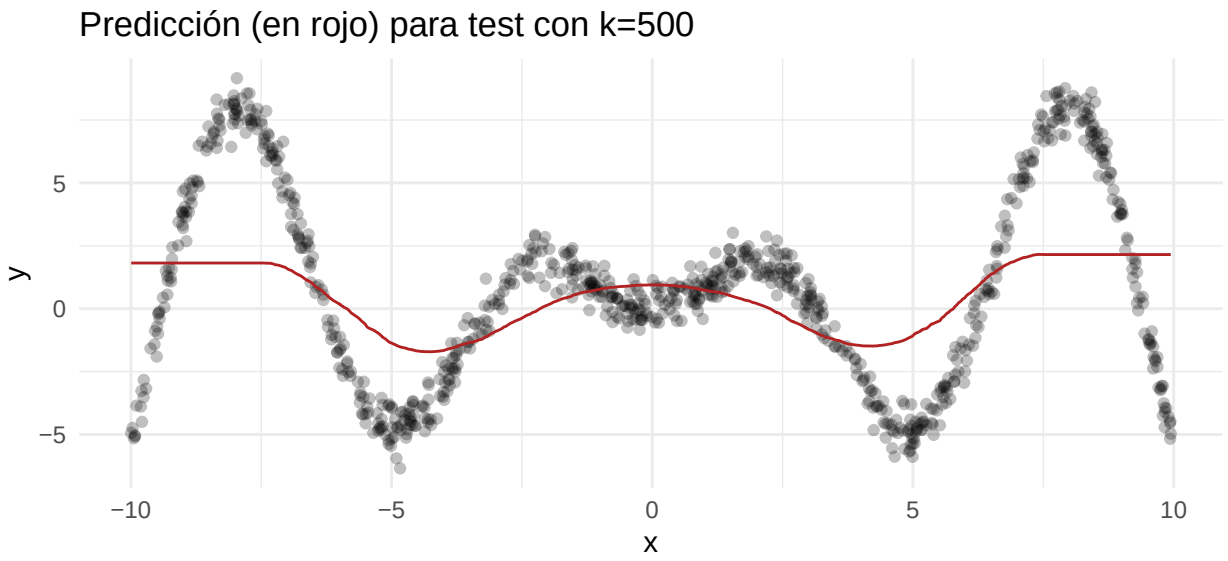
```
## [1] 2.906809
```

```
rmse(test_y, pred_test)
```

```
## [1] 3.051423
```

Como era de esperar, el RMSE en train es inferior al de test. No obstante, son valores similares por lo que no *parece* estar sobreajustando. La ventaja de tener un conjunto con dos dimensiones es que podemos representar gráficamente tanto los datos como la predicción.

```
ggplot(test, aes(x = x, y = y)) +
  geom_point(alpha = .25) +
  geom_line(aes(y = pred_test), color = 'firebrick')+
  labs(title = 'Predicción (en rojo) para test con k=500') +
  theme_minimal()
```

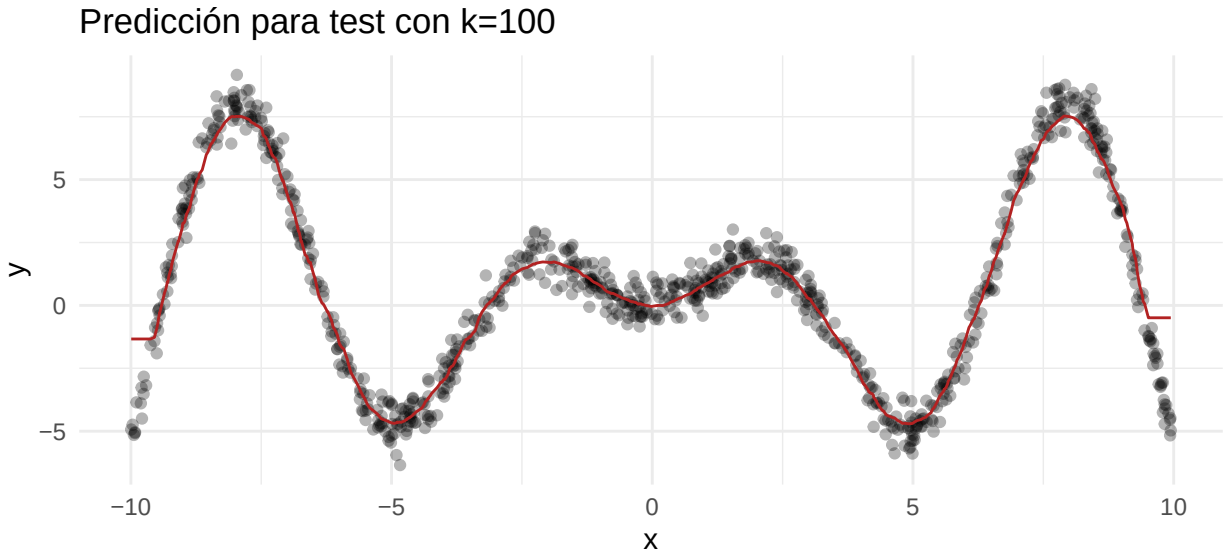


En el gráfico vemos que la predicción no es muy buena: tenemos un modelo demasiado simple (poco flexible) que no es capaz de captar la tendencia de los datos. EN un KNN esto suele significar que tenemos un valor de k demasiado alto para nuestro problema. Probemos con un valor de $k = 100$.

> **Nota:** como este proceso lo vamos a tener que realizar varias veces, el script `aux.R` incorpora una función `info_knn()` que ya hace todos los pasos necesarios para calcular la predicción, calcular los errores y representar gráficamente la predicción.

```
info_knn(k = 100,
         train_x, train_y, test_x, test_y)
```

```
## RMSE en entrenamiento:  0.7928384
## RMSE en test:  0.8202831
```



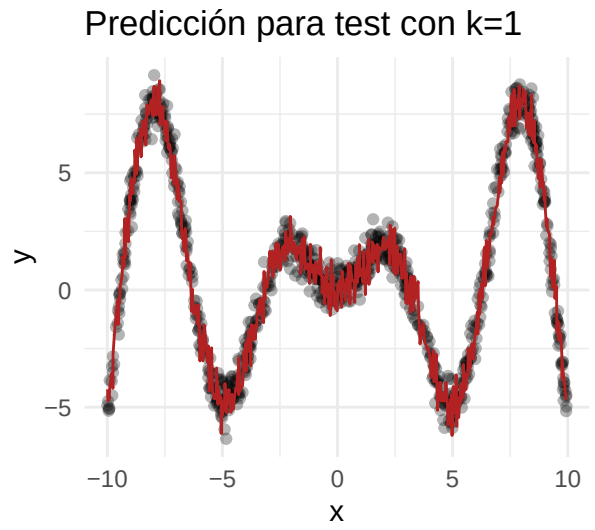
Esta vez la tendencia que sigue la predicción se parece más al verdadero patrón que siguen los datos. No obstante, fíjate que a la izquierda y a la derecha del todo la predicción no es buena.

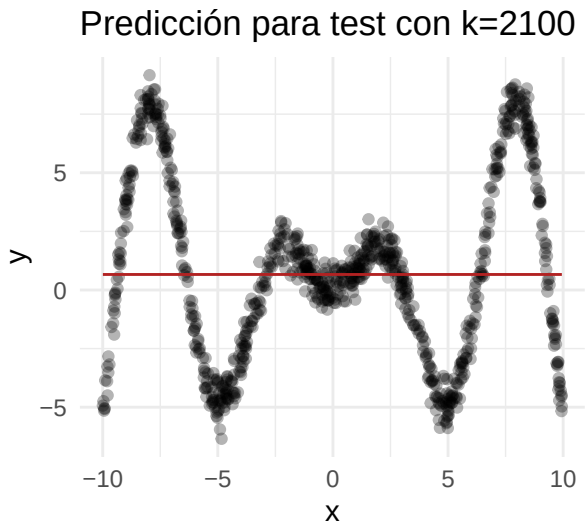
El juego que debemos seguir es el de probar con distintos valores de k para encontrar aquél que mejor encaja para nuestros datos. Antes de probar con muchos valores vamos a probar con dos casos extremos:

- $k = 1$: para predecir una observación se utiliza **solamente su primer vecino más cercano**,
- $k = n$: para predecir una observación se utilizan **todas las observaciones de entrenamiento**.

```
info_knn(1,
         train_x, train_y, test_x, test_y)

info_knn(nrow(train_x),
         train_x, train_y, test_x, test_y)
```

```
## RMSE en entrenamiento:  0
## RMSE en test:  0.7260208

## RMSE en entrenamiento:  3.650825
## RMSE en test:  3.823052
```





¿Qué puedes decir de los gráficos anteriores? Prueba otros valores para k.

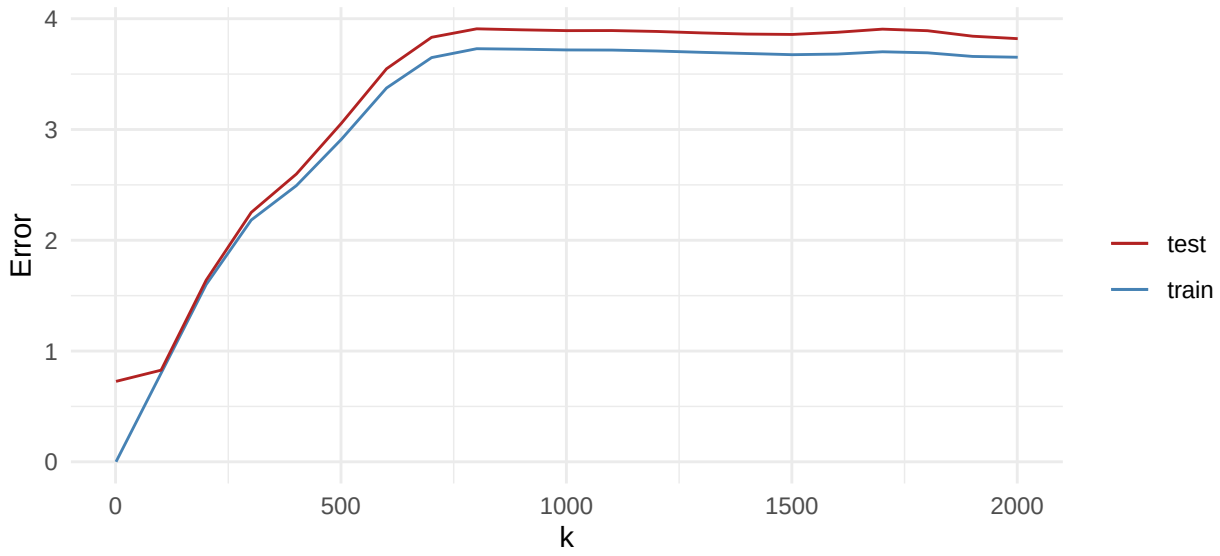
4 Búsqueda del mejor valor de k

La búsqueda del mejor valor de k para un KNN es fácil porque solo tiene un hiperparámetro: simplemente hay que calcular el error para distintos valores de k y quedarnos con aquél que menor error en test tiene.

Para los conjuntos de datos que tenemos, el valor de k puede ser desde 1 hasta 2100 (el número de observaciones en train). Sin embargo, puede llevar algo de tiempo calcular el modelo para cada valor de k por lo que es recomendable empezar con pocos valores dispersos y después ir afinando la búsqueda.

Desde el punto de vista de R, la búsqueda se realizará mediante un bucle. Para ahorrar tiempo, este bucle ya está implementado en la función `evaluation()` en el script `aux.R`. Para ejecutar la función necesitamos especificar qué valores de k queremos explorar.

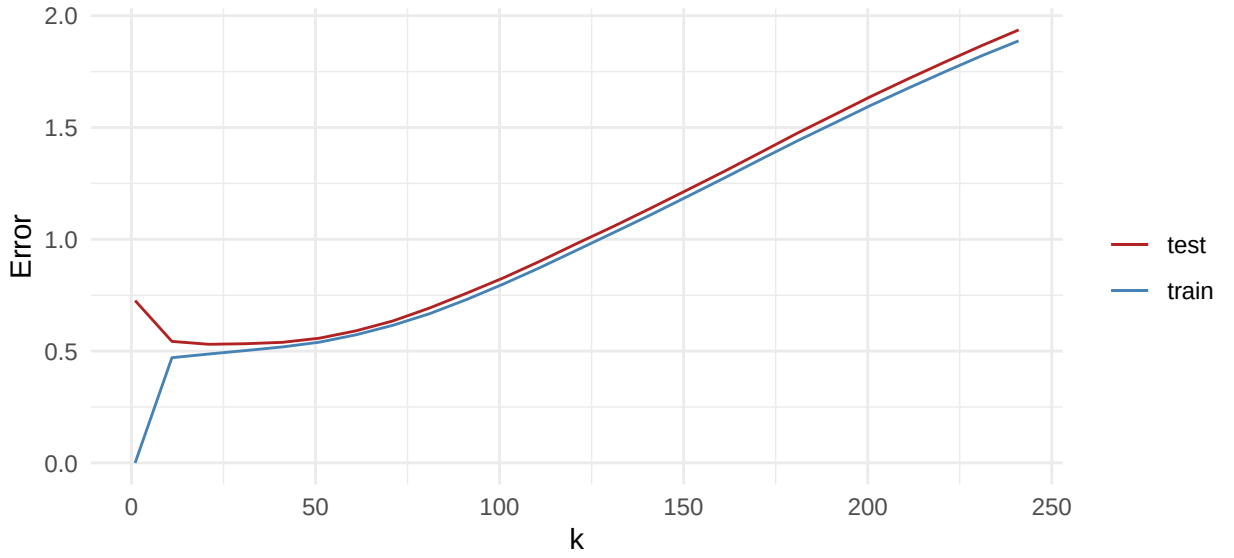
```
# Desde 1 hasta 2100 de 100 en 100
K <- seq(1, nrow(train), by = 100)

result_eval <- evaluation(K,
                          train_x, train_y,
                          test_x, test_y
                          )
```



Explora qué contiene `result_eval`

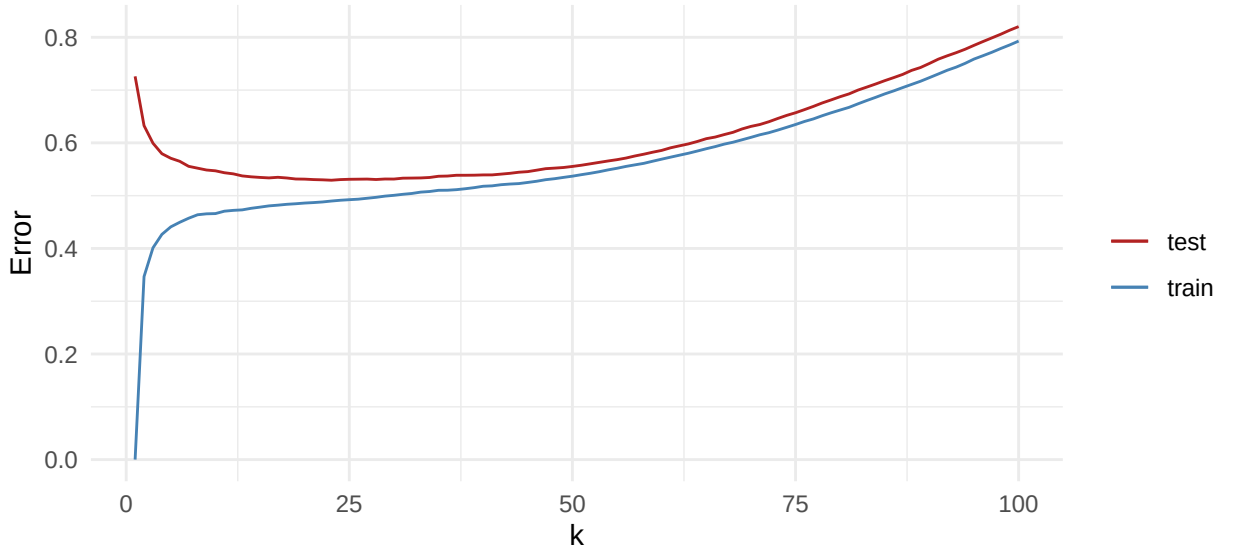
A partir de estas curvas podemos afinar la búsqueda en la región $(0, 250]$:

```
K <- seq(1, 250, by = 10)

result_eval <- evaluation(K,
                          train_x, train_y,
                          test_x, test_y
                          )
```



Está claro que el mejor valor de k debe estar en $[1, 100]$. En esta región podemos hacer la búsqueda *fina*, es decir, probando con todos los valores posibles de k

```
K <- 1:100

result_eval <- evaluation(K,
                          train_x, train_y,
                          test_x, test_y
                          )
```



En el gráfico se ve claramente el concepto de sobreajuste. Para un valor de k inferior a 25 (aproximadamente) el modelo comienza a sobreajustar lo que se traduce en que el error en train disminuye rápidamente mientras que el de test *rebota* y comienza a subir.

Podemos calcular el mejor valor para k con

```
result_eval[result_eval$rmse_test == min(result_eval$rmse_test),]
```

```
##     k rmse_train rmse_test
## 23 23  0.4898426 0.5292369
```