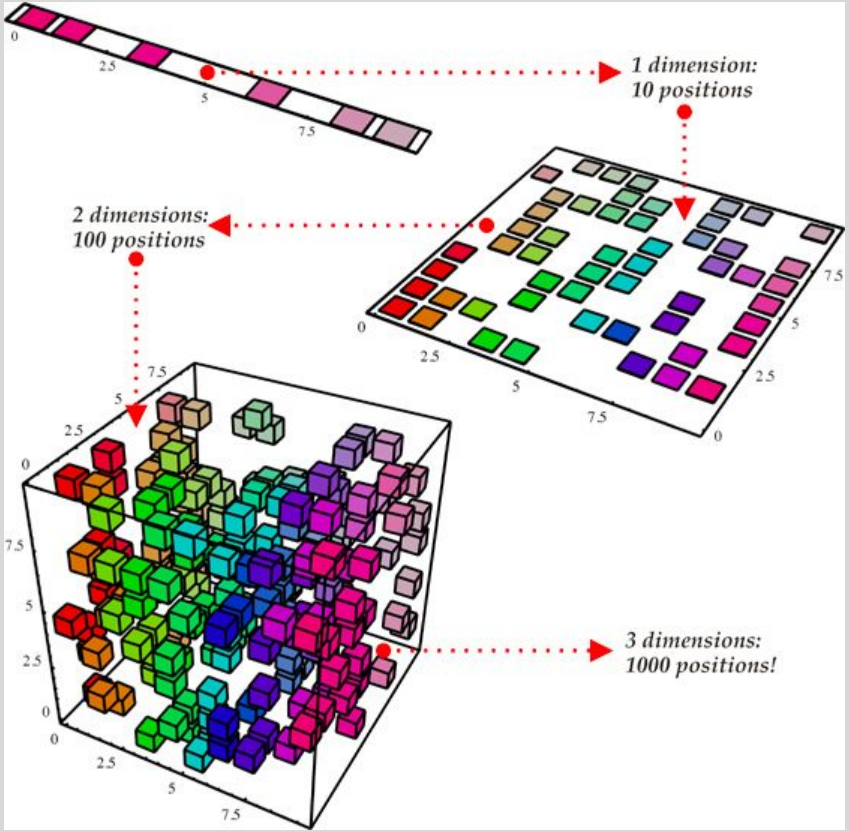


DigitalHouse > Coding School

DATA SCIENCE

UNIDAD 2
MÓDULO 4

Dimensionalidad / Clases desbalanceadas

Octubre 2017

1. **Entender cómo afecta la incorporación de muchos features la resolución de un problema de clasificación**

2. **Adquirir herramientas para resolver un problema de clasificación de clases desbalanceadas**

La maldición de la dimensionalidad

- El problema de la “maldición de la dimensionalidad” se refiere al problema de encontrar “estructura” dentro de datos en muchas dimensiones

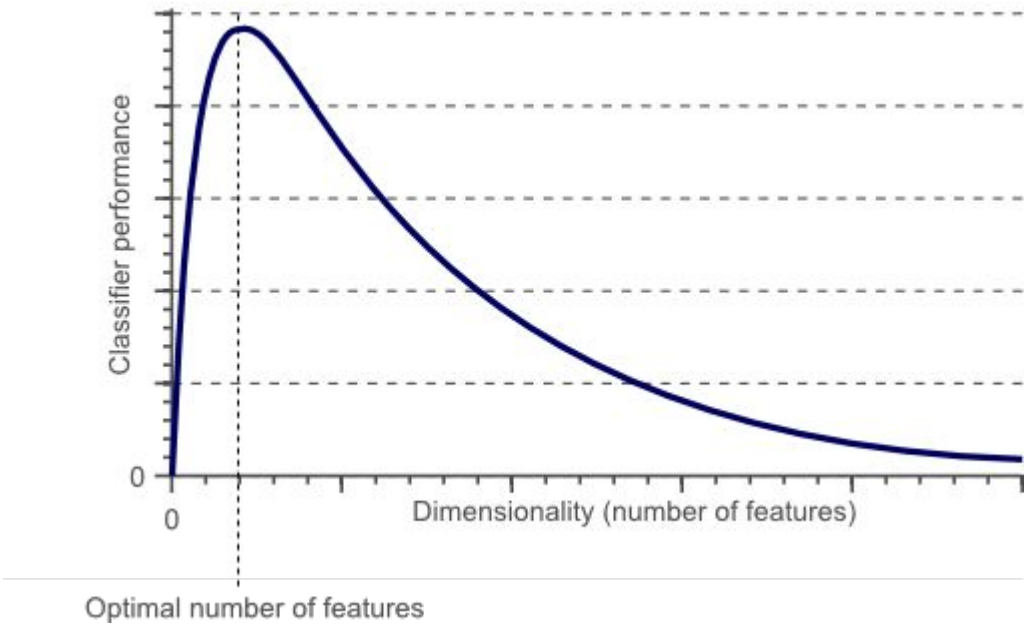
- A priori, podría pensarse que una mayor cantidad de features es mejor… no necesariamente

- Problema de clasificación: determinar si una imagen es un perro o un gato
- Features: promedio de cada color (RGB) de todos los pixels de una imagen
  - Rojo
  - Verde
  - Azul
- Clasificador de imágenes
  - IF 0.5 * rojo + 0.33 * verde + 0.2 * azul > 0.6 => gato

    ELSE perro;
- ¿Por qué no agregar más features? Saturación, tonos de grises, histogramas de color, etc.
- Es más, ¿por qué no agregar 100 o 200 de estos features para tratar de asegurarnos un clasificador perfecto?

Ejemplo

- En general, al agregar features en un problema el error mejora hasta cierto punto.
- Pero a partir de ese umbral, el clasificador comienza a empeorar su performance

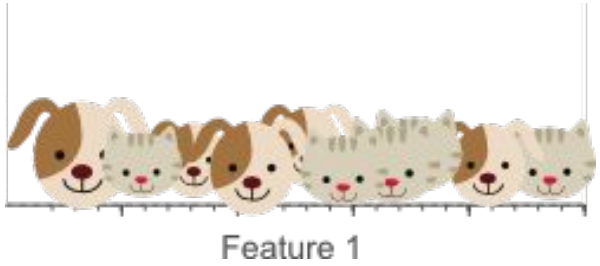


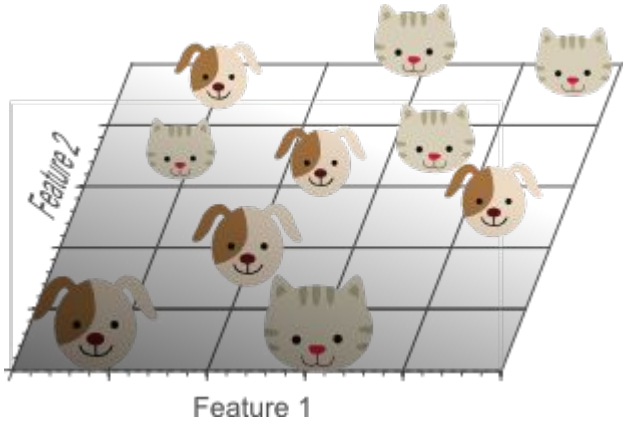
- Tenemos diez imágenes de gatos y perros y tenemos que construir un clasificador en base a estas imágenes que sirva para clasificar cualquier imágen de perros o gatos

- Usemos un clasificador lineal que haga la discriminación de forma lineal.

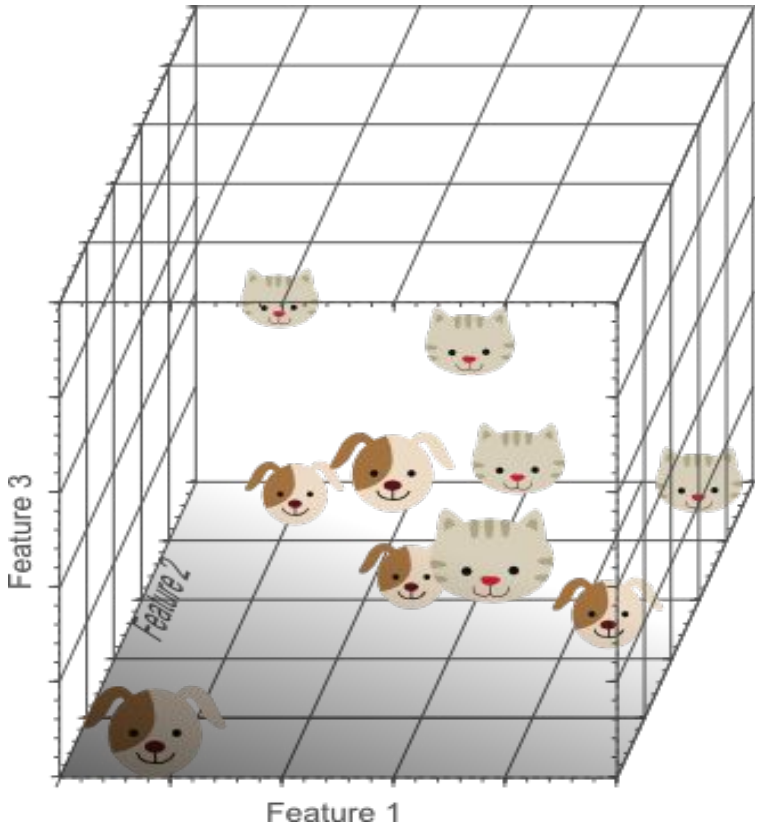
- Si usamos el feature “rojo” nuestro clasificador no funciona bien

- Agregamos la variable “verde” y no parece observarse la posibilidad de un clasificador lineal

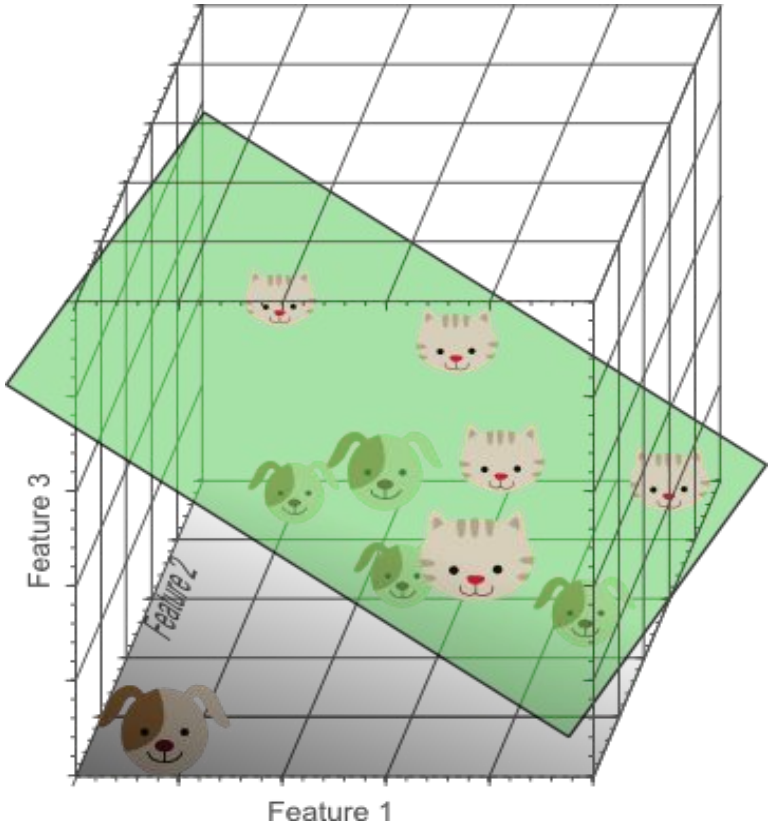




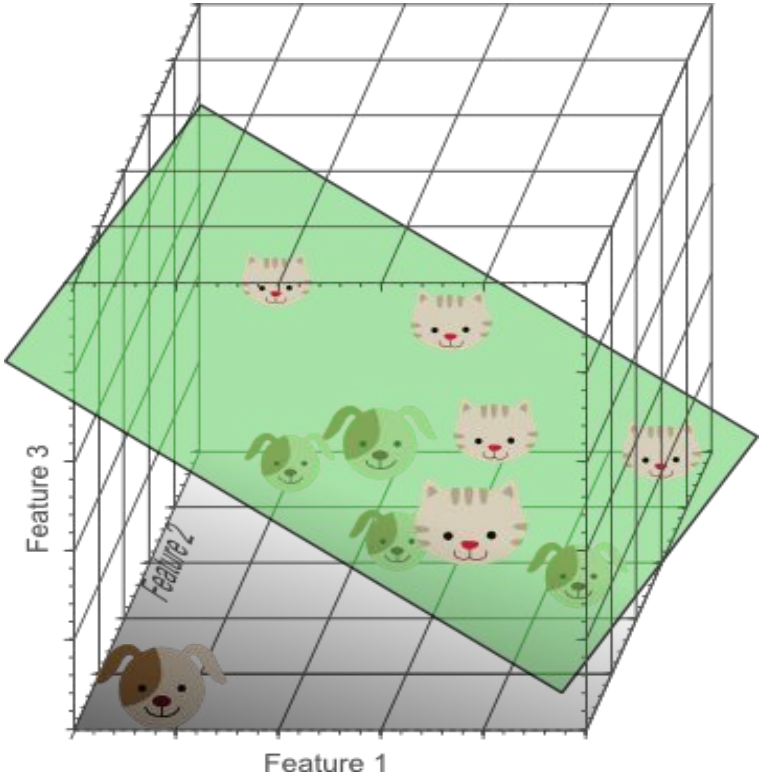
- Si agregamos una tercera dimensión (por ejemplo, rojo) la cosa parece mejorar



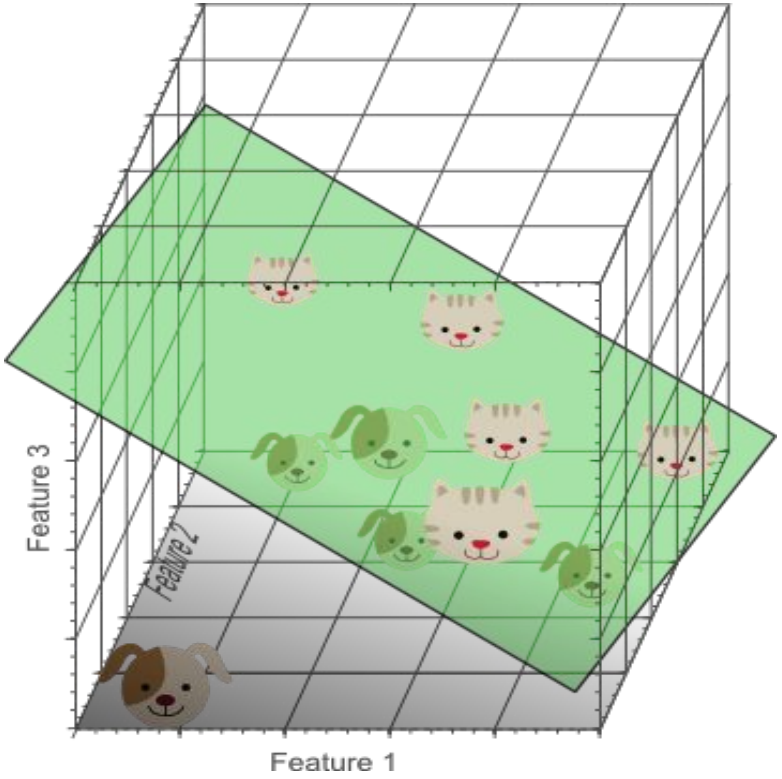
- Podemos encontrar un plano que separa perfectamente gatos y perros y que es la combinación lineal de “rojo”, “verde” y “azul” en cada pixel
- Hasta acá pareciera que agregar features mejora la capacidad de clasificación del modelo.
- Sin embargo, notar cómo la densidad de las observaciones fue descendiendo al ir agregando features.



- **Caso 1:** los 10 casos cubrían el espacio de predictores que tenía una longitud de 5. => 10 / 5 = 2 muestras por intervalo
- **Caso 2**: tenemos 10 casos (igualmente) pero en un espacio de 5 x 5 (25 posiciones) => 10/25 = 0.4 muestras / intervalo.
- **Caso 3**: 10 muestras en un espacio de 5 x 5 x 5 => 10 / 125 = 0.08 muestras / intervalo

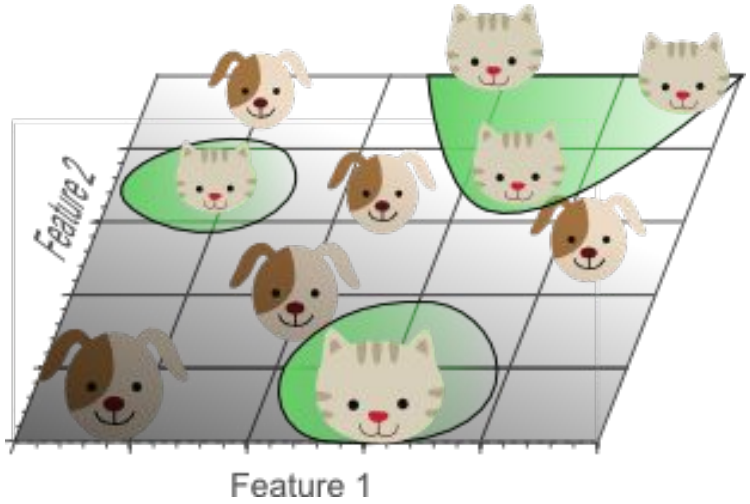


- Al agregar features la dimensionalidad del espacio de predictores se incrementa y los datos se hacen cada vez más dispersos.
- Debido a esta dispersión se hace cada vez más fácil encontrar un hiperplano que separe a las clases porque la probabilidad de que un dato de training se encuentre del lado equivocado del hiperplano se hace infinitamente pequeña
- Proyectemos este hiperplano a las dos primeras dimensiones



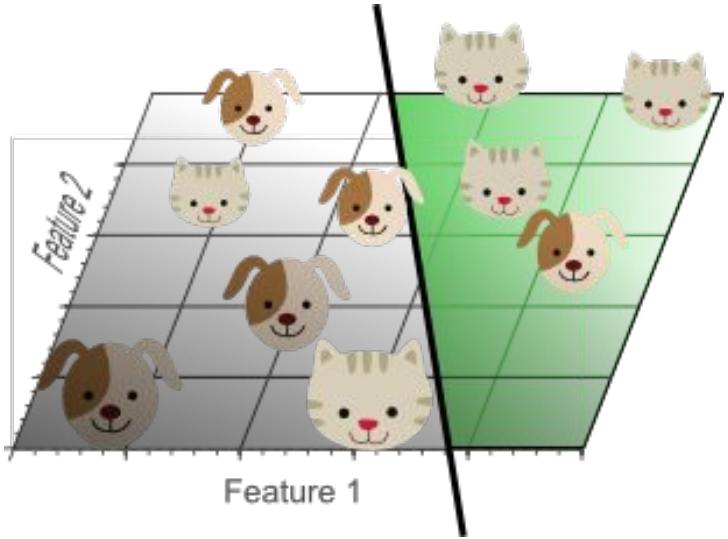
- Se ve que entrenar el clasificador con tres features es equivalente a entrenar un clasificador extremadamente complejo en un espacio de menor dimensionalidad.

- Entonces corremos el riesgo de caer en el **overfitting**.

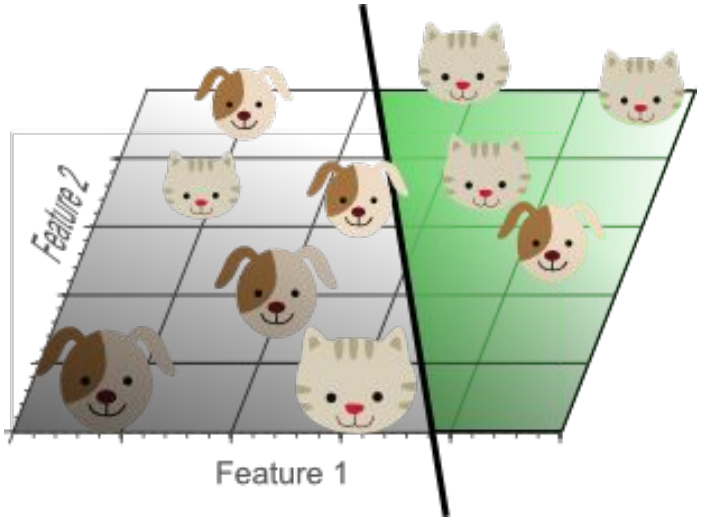


- Un clasificador lineal en dos dimensiones (aunque más simple) es más generalizable a casos “nuevos” que el clasificador lineal en tres dimensiones.

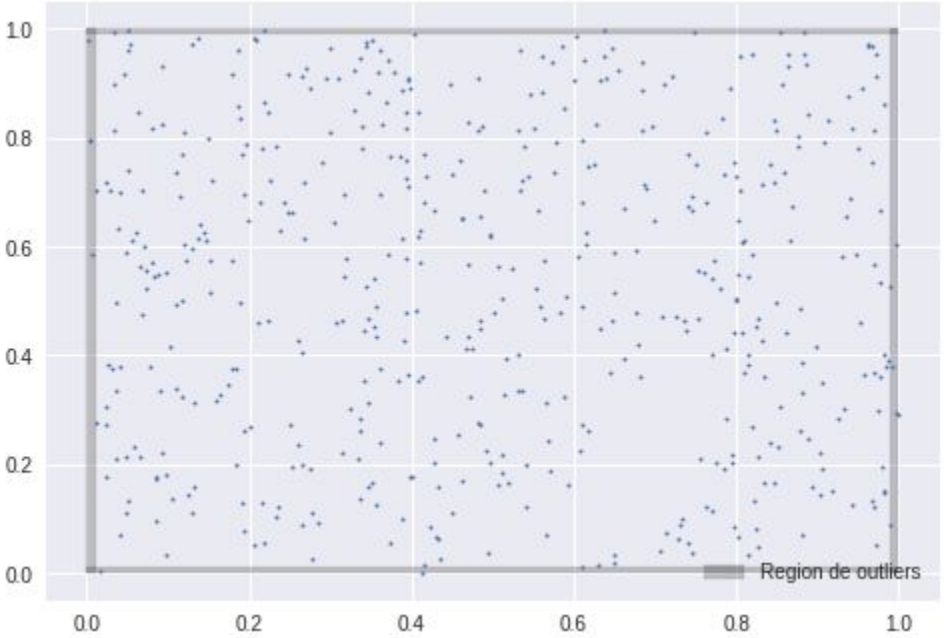
- En otras palabras: en muchos casos, al usar una menor cantidad de features podemos obtener un modelo con mayor capacidad de generalización, dado que se se ve potencialmente menos afectado por el overfitting



- Entonces, a mayor cantidad de features (e igual cantidad de casos) se incrementa la dispersión de los casos en el espacio de predictores,
- Además, esa mayor dispersión no se distribuye de forma homogénea en el espacio.
- En general, al incrementarse la dimensionalidad del problema, cada vez más muestras se ubican en los extremos del espacio, por lo cual, se hace más difícil construir un buen clasificador.

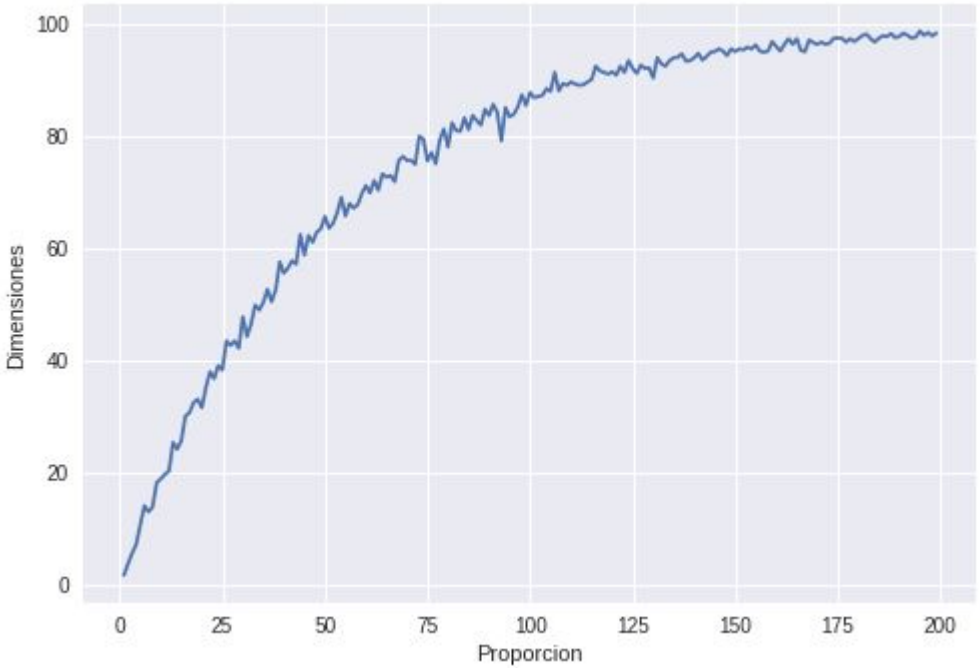


- Si tenemos una variable distribuida uniformemente en un hipercubo, ¿qué proporción de outliers encontramos?
- Podemos definir outliers como aquellos puntos que toman valores extremos en alguna de las “d” dimensiones.

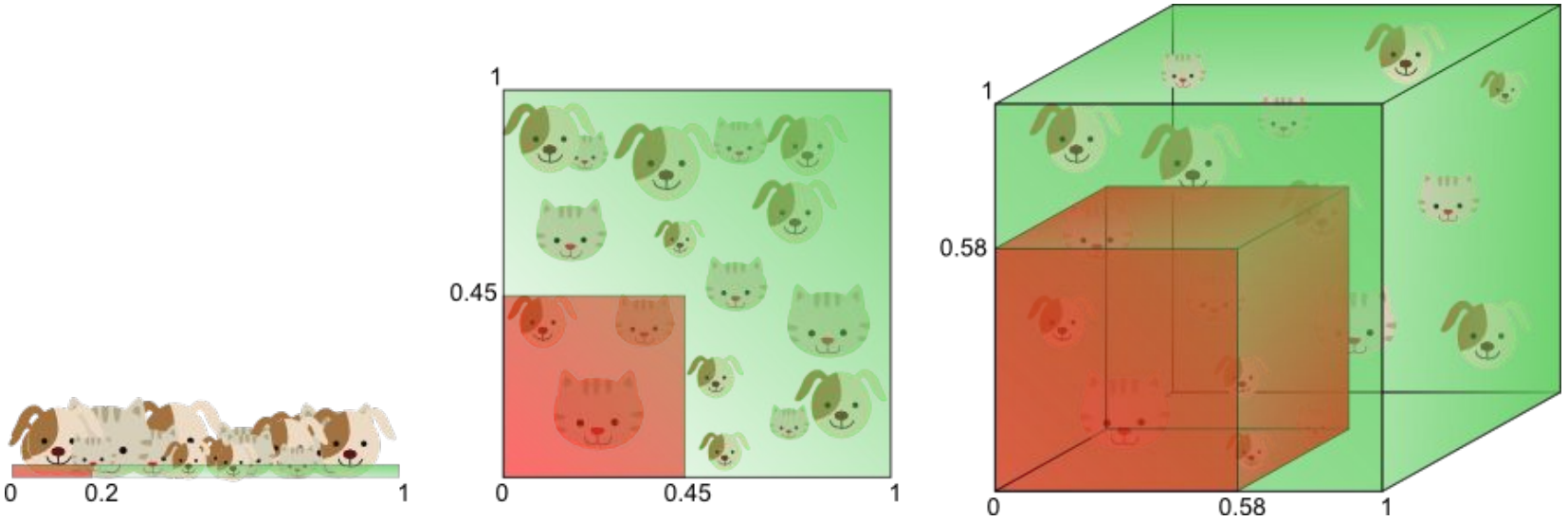


- Ahora bien, ¿qué pasa a medida que aumentamos la dimensionalidad del dataset?

- Dado que no podemos graficar en más de tres dimensiones, lo que haremos será graficar la evolución de la proporción de outliers para cada nivel de dimensionalidad.



- En el ejemplo de gatos y perros la cosa se vería así:



- En general, cuanto menor sea la cantidad de casos de entrenamiento, menor será la cantidad de features que podrán usarse
- Uso de algoritmos de selección de features (lasso, best subset selection, forward selection, backward selection, etc.)
- Uso de algoritmos de reducción de dimensionalidad (clustering, PCA, etc.)

Datasets desbalanceados

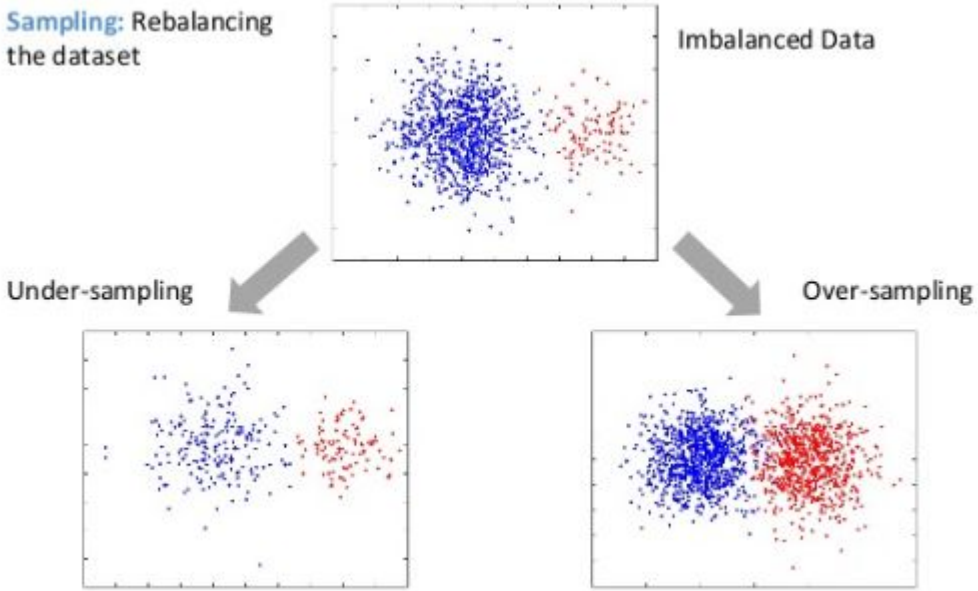
- Los datasets del mundo real suelen presentar clases desbalanceadas.

- ¿Qué pasa en un dataset desbalanceado con el accuracy?

  El accuracy suele ser muy alto, pero esto generalmente es porque el modelo responde la “solución trivial” o algo muy cercano.

- Algunos casos comunes de datasets desbalanceados son:
  - Estudios sobre fraude
  - Diagnóstico de enfermedades

- Una de las técnicas para combatir el desbalance del dataset es el resampling: repetir los casos de la clase minoritaria o descartar algunos casos de la clase mayoritaria.
- Estas técnicas se conocen como **oversampling** y **undersampling**



Existe un paquete de Python específicamente diseñado para abordar el problema del oversampling: imbalanced-learn. El paquete no forma parte de sklearn, pero adopta sus estándares para ser compatible con los demás modelos.

Implementa, entre otras técnicas SMOTE: Synthetic Minority Oversampling Technique.

Consiste en los siguientes pasos:

1. Se elige un punto al azar de la clase minoritaria y sus K vecinos más cercanos.
2. Se elige al azar uno de esos vecinos.
3. Se calcula el vector entre el punto seleccionado y el vecino seleccionado al azar y se lo multiplica por un número aleatorio entre 0 y 1.
4. El punto aleatorio dentro del vector es el nuevo dato para el oversampling.

- Otra técnica es tomar en cuenta el desbalance en la función de costos del algoritmo.
- Algunos algoritmos de SKlearn tienen implementada la posibilidad de incluir pesos en su función de costos. Por ejemplo:
    - SVM
    - Regresión Logística
- La matriz de pesos también puede ser una decisión de implementación que conviene explorar utilizando cross validation.

Práctica Guiada: Clasificación con clases desbalanceadas

- El problema de las clases desbalanceadas en tareas de clasificación debe ser tenido en cuenta dado que afecta la performance de/los modelo/s utilizados

- Es necesario tener en cuenta que no debería agregarse de forma indefinida features a los modelos dado que se incrementa la dimensionalidad del problema y junto con ello, el riesgo de overfitting.