

Clase 2: 13 de junio de 2019

Fundamentos de Machine Learning.

Parte 2 de 2

EAE 253 B

C Dagnino. cdagnino@gmail.com

Resumen I

$$Y = f(X) + \varepsilon$$

- $f(X)$ puede ser una función muy compleja (no tiene por qué ser lineal)
- El objetivo es predecir Y (en el futuro).
- ε son factores desconocidos, así que lo más razonable es aproximar $f(X)$
- A la estimación la llamamos $\hat{Y} = \hat{f}$
- La evaluación se realiza mirando Y vs \hat{Y}

Resumen II

- Un buen modelo predictivo encuentra el compromiso preciso entre sobreajuste (overfit) y subajuste (underfit)

Subajuste	Sobreajuste
mucho Sesgo	muchas varianzas
muy poca complejidad	demasiada complejidad
alto error in-sample	bajo error in-sample
alto error out-of-sample	

Descomposición fundamental I: Sesgo vs Varianza

Error esperado al cuadrado = $E[(y - \hat{f})^2]$

$$= Sesgo[\hat{f}]^2 + Var[\hat{f}] + Var[\varepsilon]$$

- $Var[\varepsilon]$ se llama error irreducible
- Sesgo y Varianza son dos tipos de error; el truco es reducir uno sin aumentar tanto el otro

Descomposición fundamental II: en-muestra vs fuera-de-muestra

Error Real de Predicción = error en-muestra +
complejidad del modelo

Error Real de Predicción = error en-muestra +
"optimismo"

Un poco más formal

- Nuestra base de entrenamiento es $x_{entr}, y_{entr} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$.
- Queremos predecir en nuestra base de test. Estas observaciones nuevas son (x_{test}, y_{test})
- Usamos la base de entrenamiento para estimar f y obtener \hat{f} . Por ejemplo, en una regresión lineal, esto equivaldría a obtener $\hat{\beta}_0, \hat{\beta}_1$ y luego:

$$\hat{f} = \hat{\beta}_0 + \hat{\beta}_1 x_1$$

Objetivos de minimización

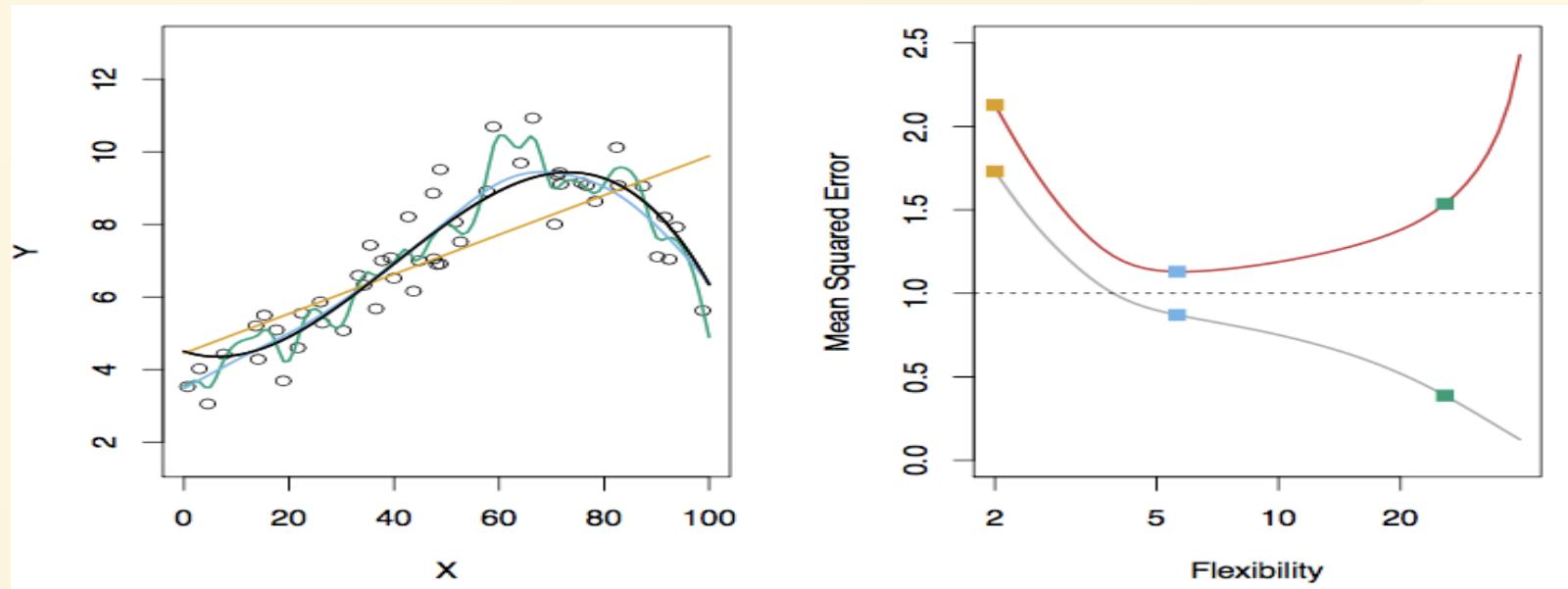
- error real de predicción =
 $Promedio(\hat{f}(x_{test}) - y_{test})$. En otras palabras:
el error en promedio que tendremos con nuevas
observaciones.
- error en muestra =
 $Promedio(\hat{f}(x_{entrenamiento}) - y_{entrenamiento})$
- Es fácil minimizar el error en muestra (al menos
con los computadores de 2019!)

¿Cómo podemos minimizar el error real de predicción?

- O: ¿cómo elegimos el modelo con el menor error real de predicción?
- Usar el error en muestra pareciera ser una buena aproximación (parecen ser lo mismo pero aplicados a una muestra diferente!)
- La gran diferencia: la base de entrenamiento es la que se usa para estimar f . Si se usa la base de test para entrenar/estimar f , deja de ser realmente una base de test.

Simulación de error para distintos grados de complejidad

Con una f conocida, se simulan datos de x e y . Se divide en base de entrenamiento y test. Luego se estiman tres modelos con la base de entrenamiento

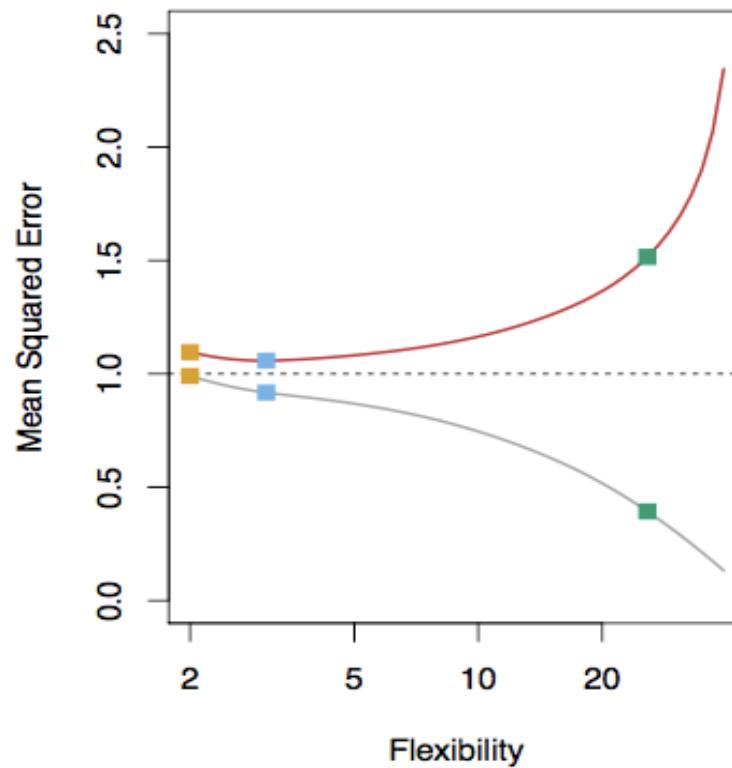
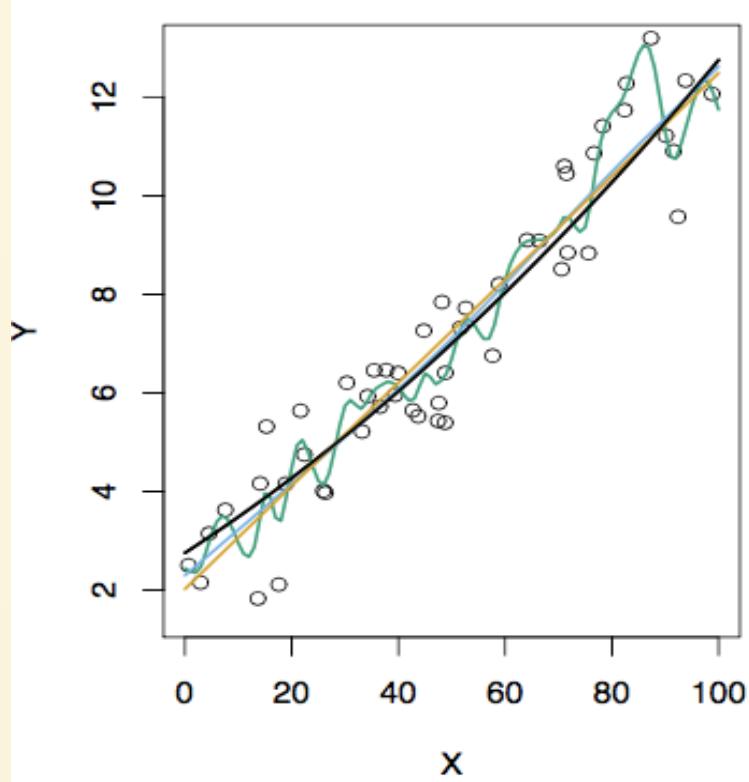


- A la izquierda: estimación con funciones cada vez más flexibles (amarilla es la lineal) o más *grados de libertad* La línea negra es la real.
- En rojo a la derecha: error de la base de test
- En negro a la derecha: error en la base de entrenamiento

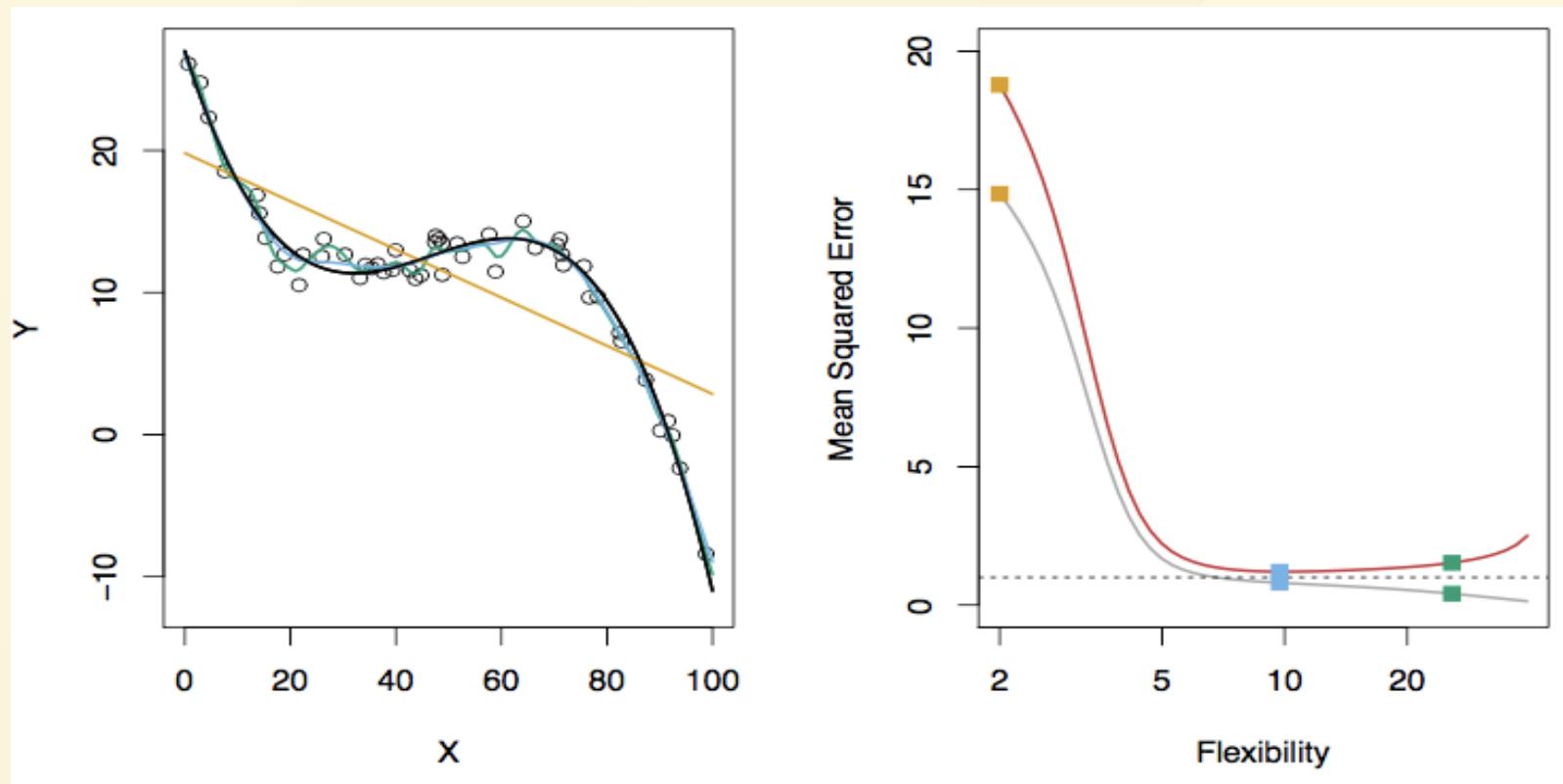
Observaciones

- La línea punteada en 1.0 es el error irreducible. No se puede ir más abajo que eso
- La línea amarilla hace subajuste!
- La línea verde hace sobreajuste: memoriza patrones de los datos de entrenamiento que en verdad se deben al azar.
- Una gran diferencia entre error de entrenamiento y de test sugiere sobreajuste

Ahora con una f casi lineal



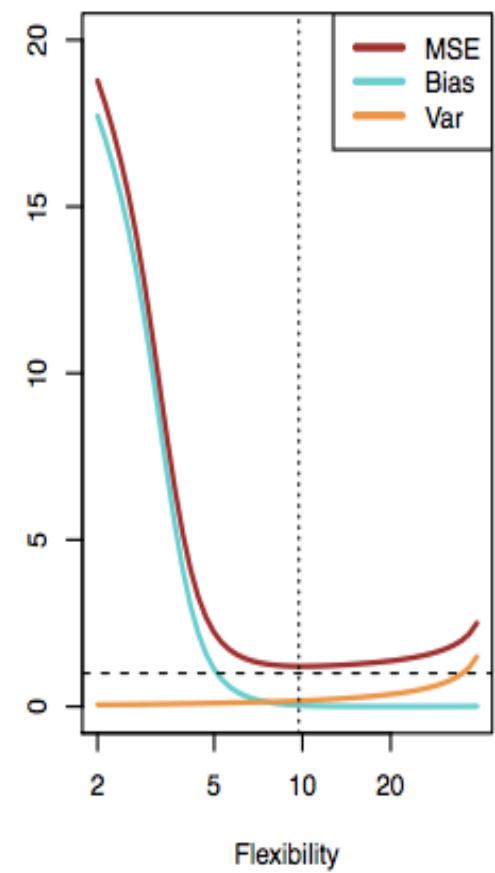
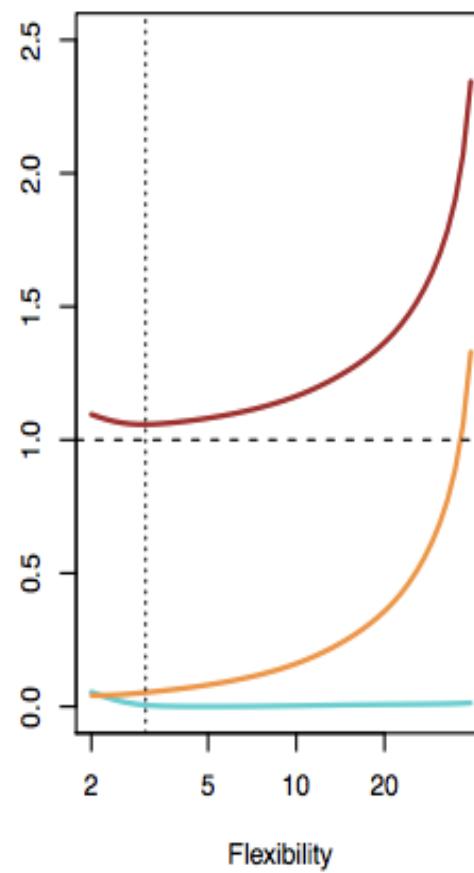
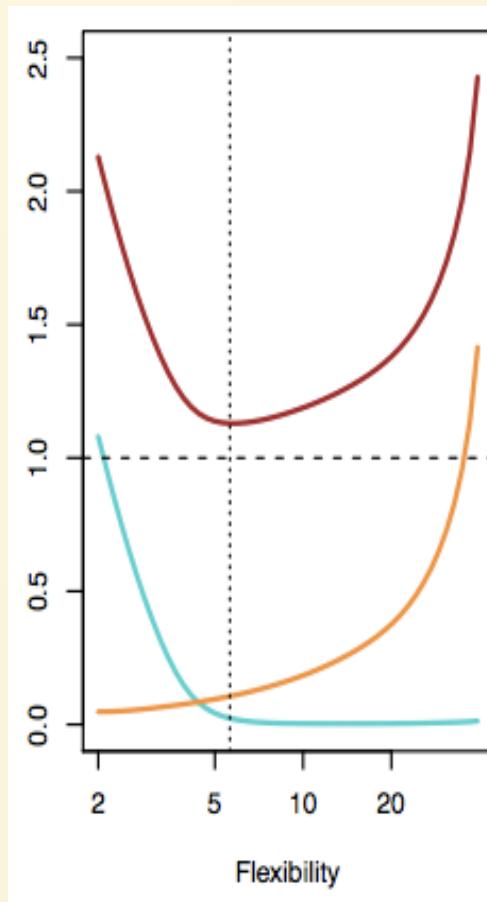
Y una muy no lineal



¿Por qué el error de test tiene forma de U?

- Compromiso entre sesgo y varianza!
- El sesgo disminuye al aumentar la flexibilidad, pero aumenta la varianza
- Ricitos de oro: disminuir el sesgo, pero no tanto: después el aumento en varianza se come la ganancia.
- Costo marginal = ingreso marginal!

Resumen: f intermedia, f lineal, f bastante no lineal

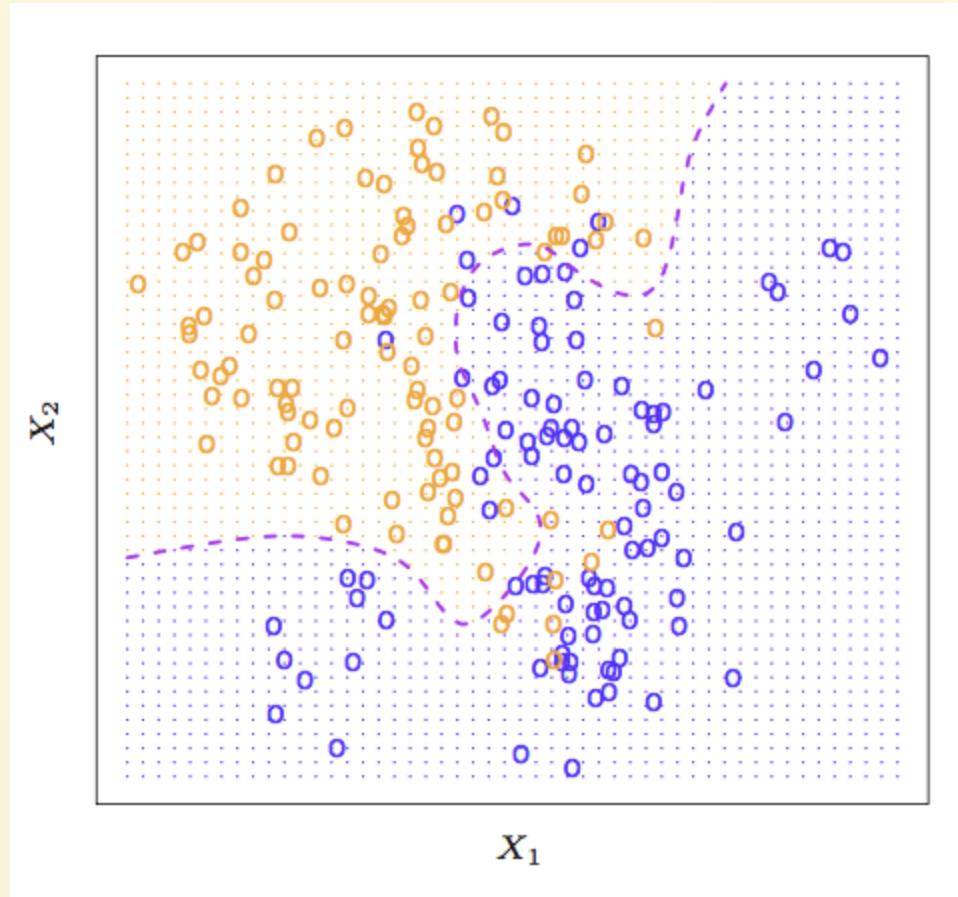


Ejemplo con clasificación

- Clasifico a y en k categoría
- El error cuadrático medio no nos sirve en este contexto
- El error se produce cuando \hat{y} no tiene la misma categoría predicha que el y real
- Una posibilidad para medir error, donde I vale uno si lo de dentro se cumple:

$$\frac{1}{n} \sum I(y_i \neq \hat{y}_i)$$

Datos simulados



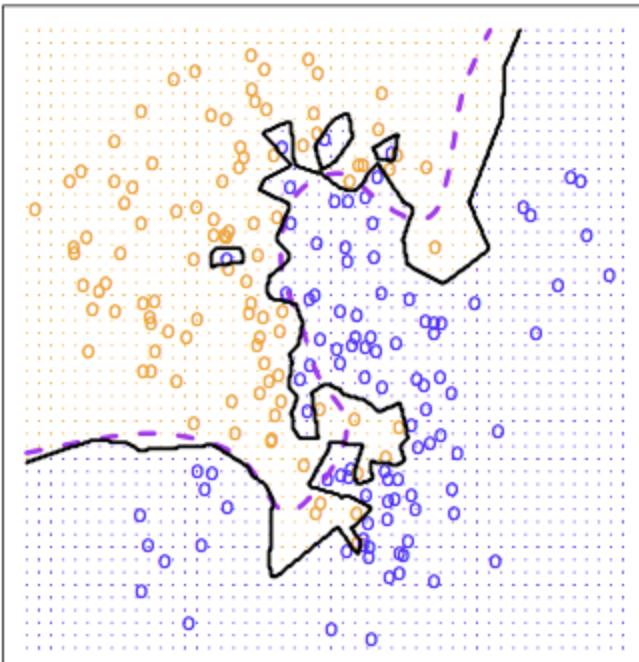
- La frontera es la correcta (clasificador de Bayes), pero hay ruido, por eso algunos morado y algunos

Un método para clasificar: K-nearest neighbors (KNN)

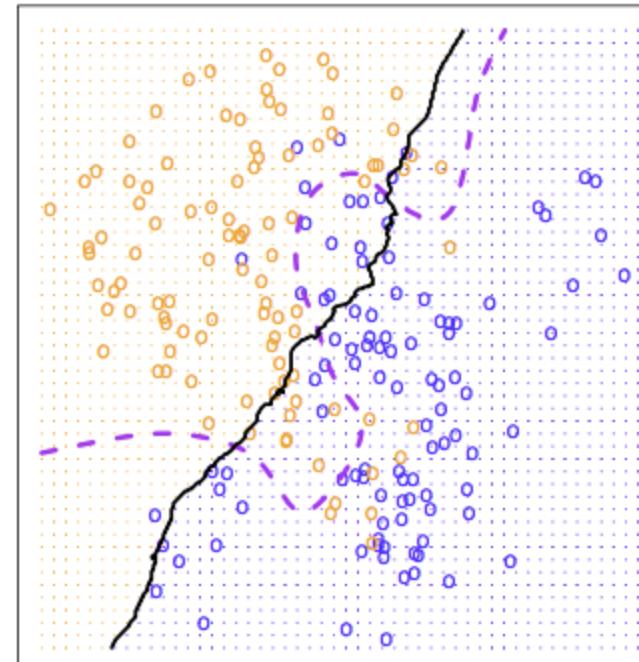
- Para cada punto de la base de test (pongamosle x_0 , identifica a las K observaciones más cercanas
- Luego estima la probabilidad de que x_0 pertenezca a los morado contando la proporción de morados entre las K observaciones más cercanas
- Mientras más pequeño es K , más flexible y menor sesgo (pero mayor varianza!).

Ricitos de Oro en clasificación

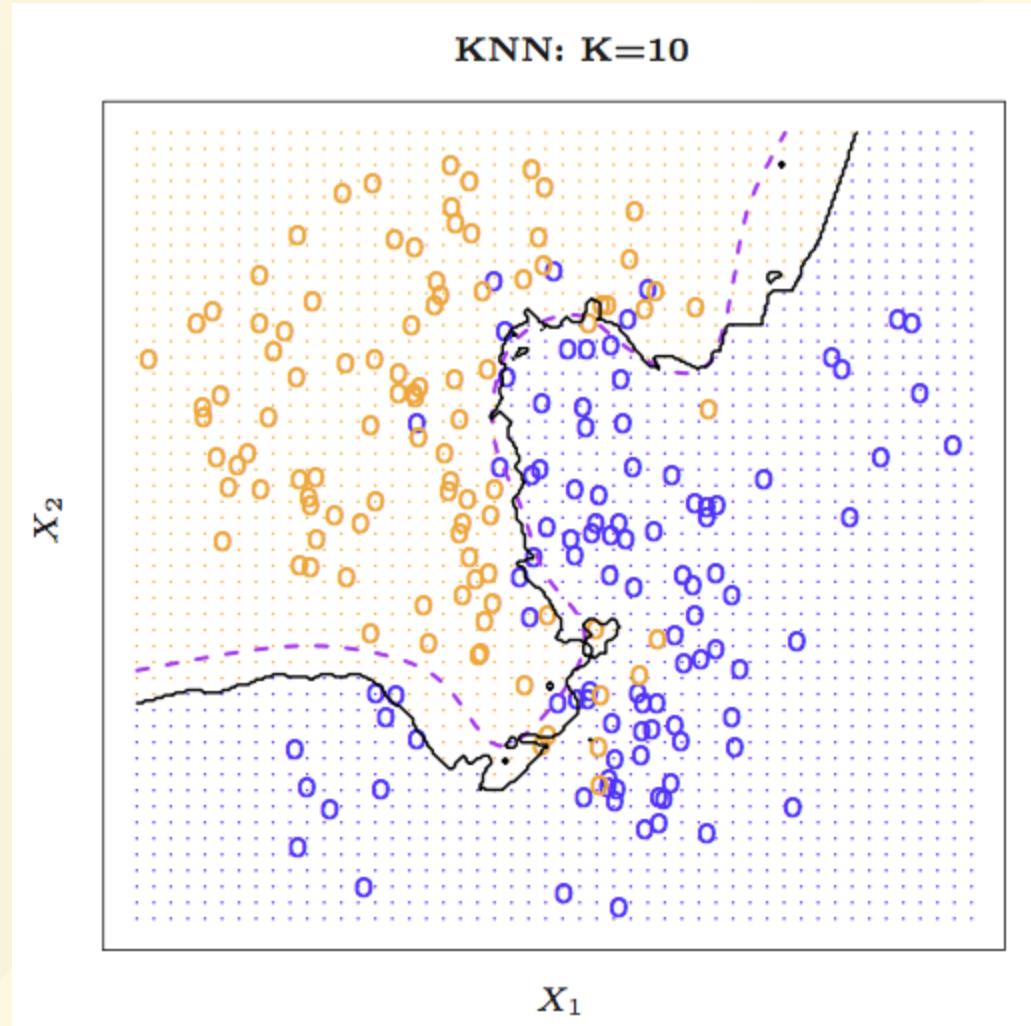
KNN: K=1



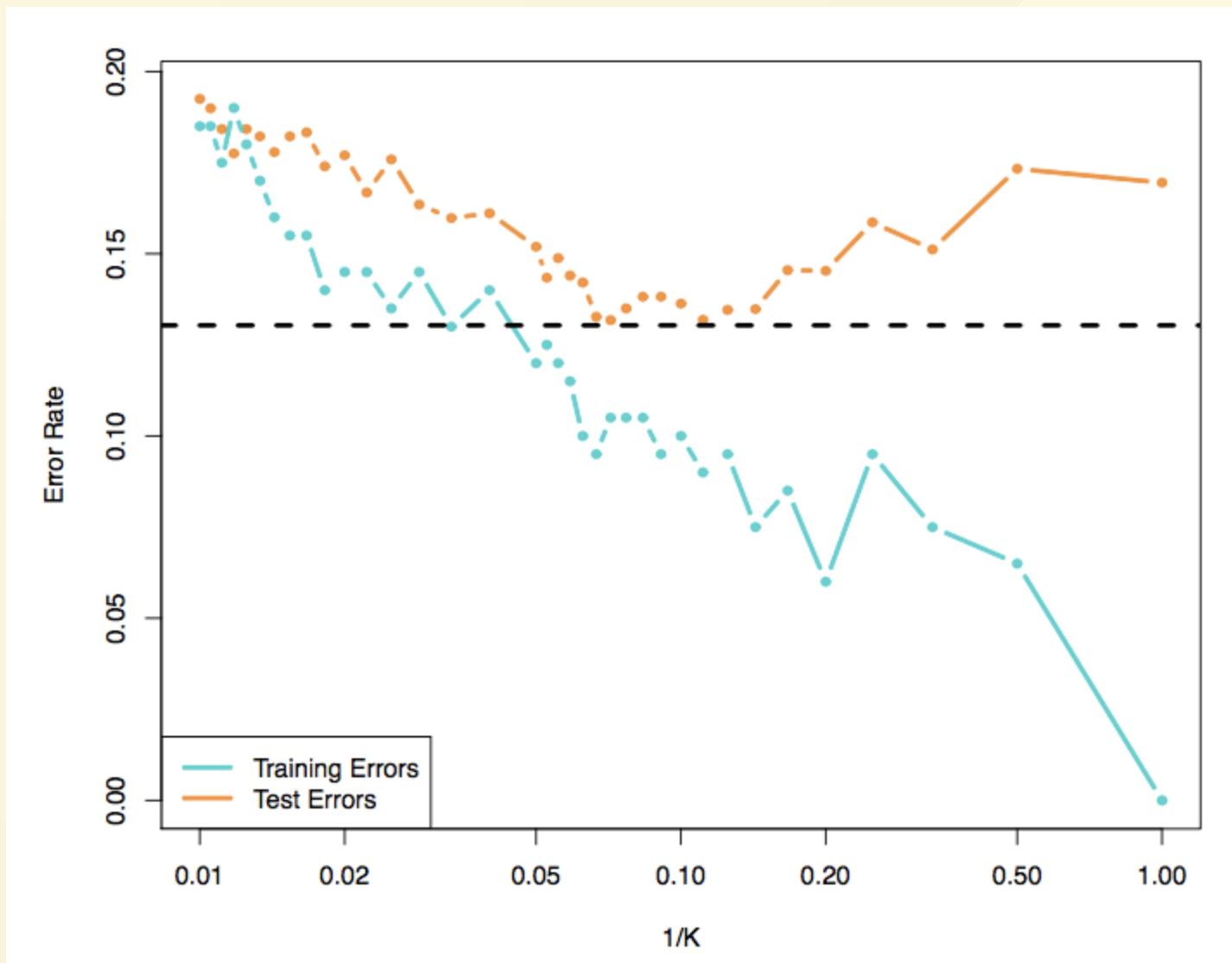
KNN: K=100



Un mejor K



Error de entrenamiento y de test



¿Cómo podemos entrenar modelos que minimicen el error verdadero?

1. Funciones de estimación que tengan un castigo por complejidad
2. *cross-validation* y selección de variables
3. Promedio de modelos para reducir sesgo, varianza / aumentar estabilidad (*boosting*: Random Forests, *bagging*).
4. Si hay suficientes datos, apartar una base de *test* para hacer la última evaluación de modelos

Estas cuatro aproximaciones no son excluyentes

1. Castigo por complejidad

$$\hat{f}_{ols} = \operatorname{argmin} \sum_{i=1}^n (y_i - f(x_i))^2$$

$$\hat{f}_{ML} = \operatorname{argmin} \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda R(f)$$

- λ es un hiperparámetro. Si es muy grande, penaliza demasiado la flexibilidad. Si es muy pequeño, penaliza muy poco la flexibilidad
- La idea es controlar la complejidad del modelo de una manera sistemática.

Ejemplos de $R(f)$

- El algoritmo de *Lasso* usa $R(f) = \sum_{k=0}^K |\beta_k|$. Con dos variables, *lasso* minimiza el siguiente problema:

$$\sum_{i=1}^n (y_i - \beta_0 - \beta_1 x_1 - \beta_2 x_2)^2 + \lambda(|\beta_0| + |\beta_1| + |\beta_2|)$$

- En el algoritmo de *KNN* (*k-nearest-neighbors*), $R(f)$ es el K , el número de vecinos considerados para estimar el modelo.

2. Cross-validation

- Entre otras cosas, permite elegir los hiperparámetros como λ
- La base de training es divide en K partes (*folds*)
- Se usan $K - 1$ partes para entrenar el modelo y se predice para la parte no usada (funciona un poco como *test*)
- Elegimos el valor de λ que minimice el error en esas partes

Referencias sobre la descomposición

- Introduction to Statistical Learning, cap. 2 (buena introducción). Para más detalles: Cap 7 (grados de libertad) y cap 5 (cross-validation)
- Elements of Statistical Learning, Cap. 7 (muy completo, pero más complejo)
- auto-promoción: [Derivación de la descomposición y grados de libertad](#)

3. Promedio de modelos

Ejemplo: Random Forests

<http://www.r2d3.us/visual-intro-to-machine-learning-part-1/>

- Conjunto de árboles de decisión poco profundos (~100, ~200)
- Se previene overfit de varias maneras:
 - Árboles tienen un máximo de ramas
 - Se usa cross-validation para medir el error
 - Un número reducido de variables por árbol

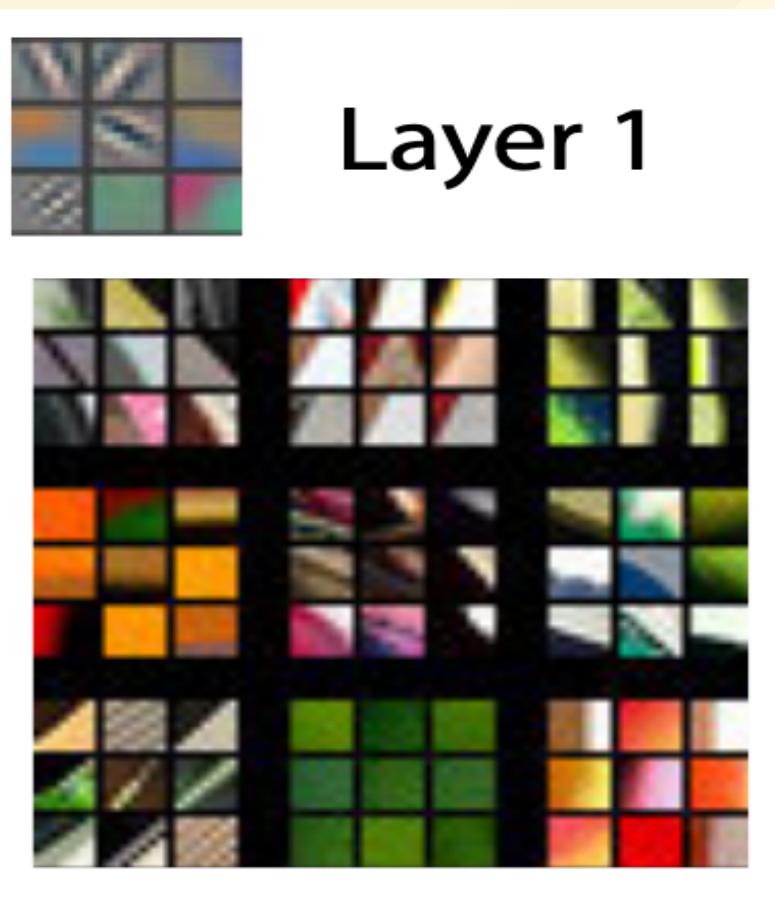
Reconocimiento de imágenes y texto

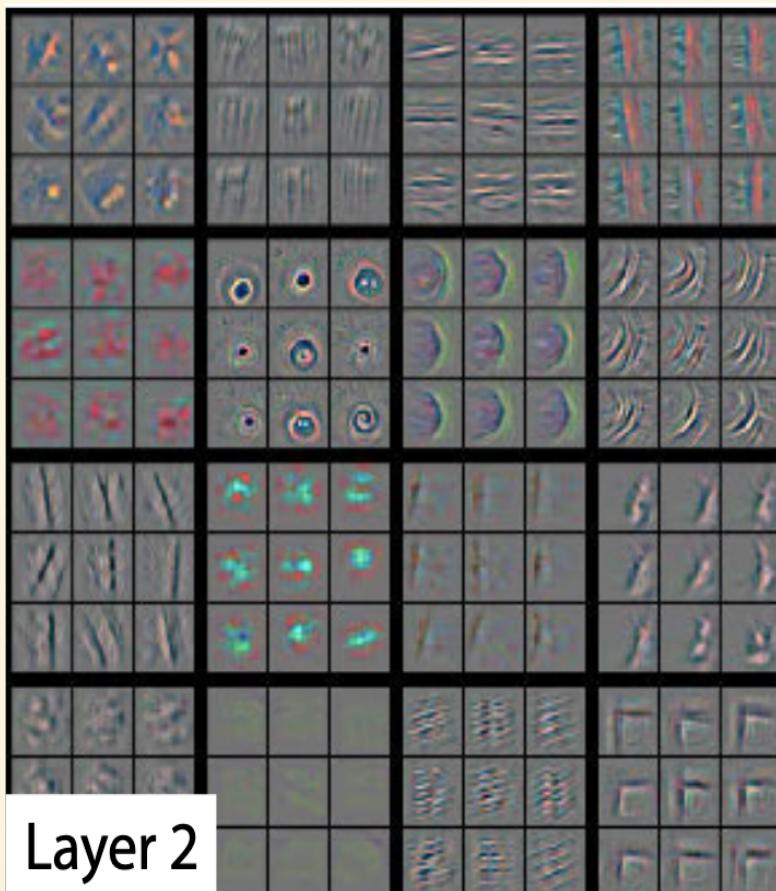
- Y puede ser una categoría de una imagen
- También puede ser el sentimiento expresado en una frase (pena, alegría, rabia)
- Tanto una imagen como un texto tienen "variables" o "features"
- El principio es el mismo: los datos de entrenamiento permiten al modelo aprender patrones que se correlacionan con cómo se ve un gato

Reconocimiento de imágenes con redes neuronales

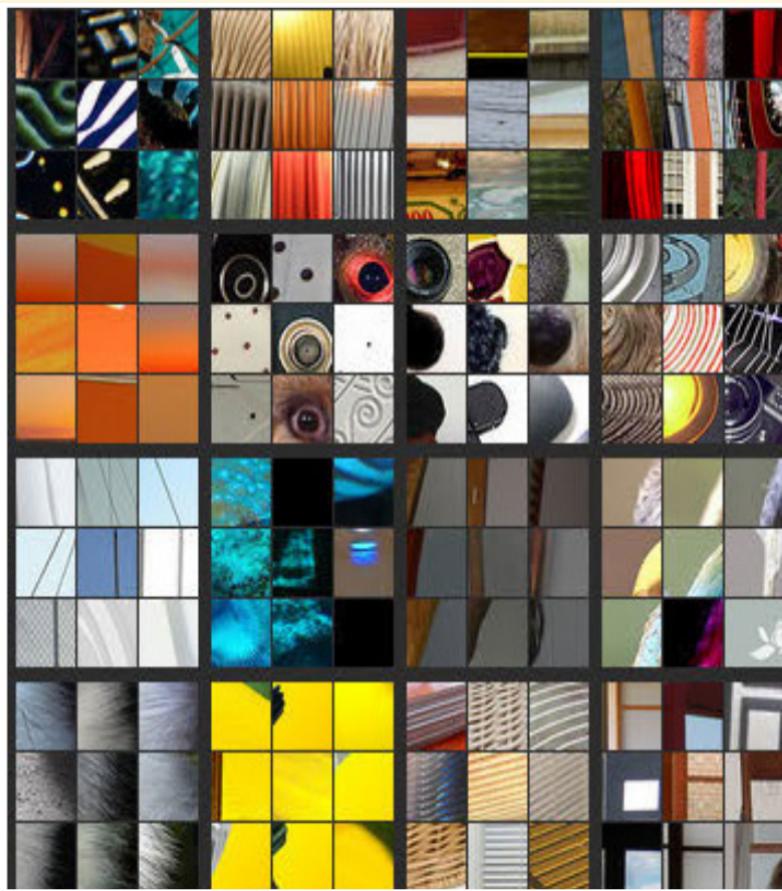
- Y es la etiqueta de la imagen: gato, perro, silla
- X son los pixeles
- $f(x)$ puede ser arbitrariamente complejo: las redes toman el input inicial de una imagen (una matriz de pixeles), pero luego generan interacciones y funciones de estas x , para generar cosas más complejas

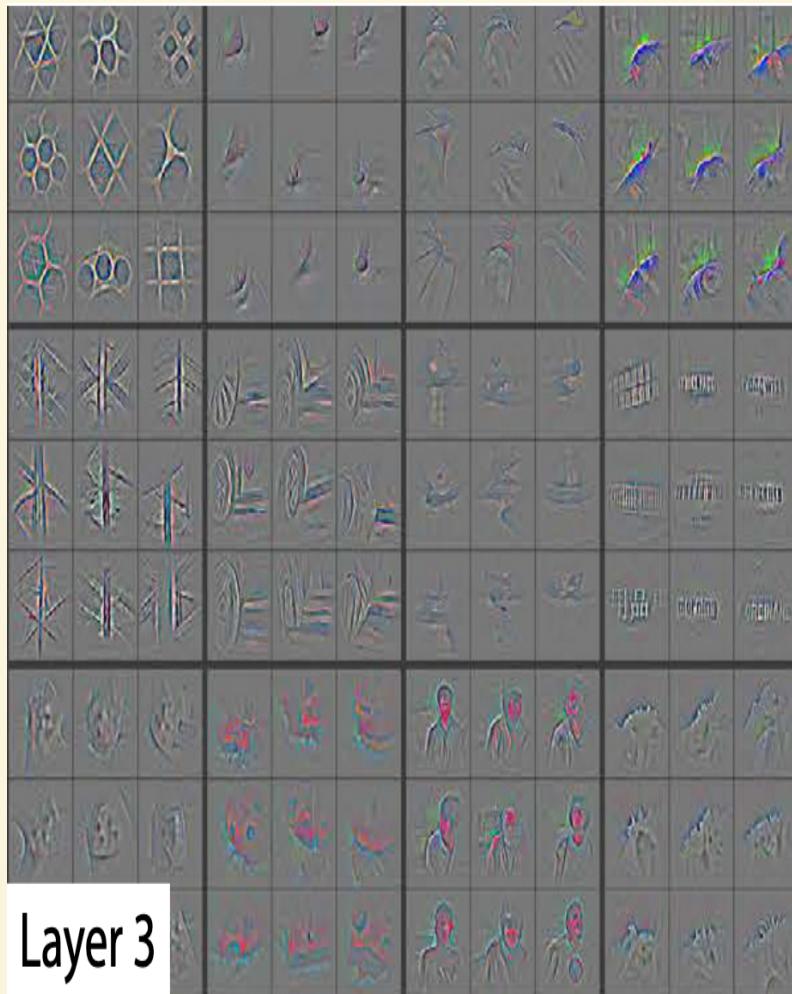
Avanzamos en las capas y los X se vuelven más complejos





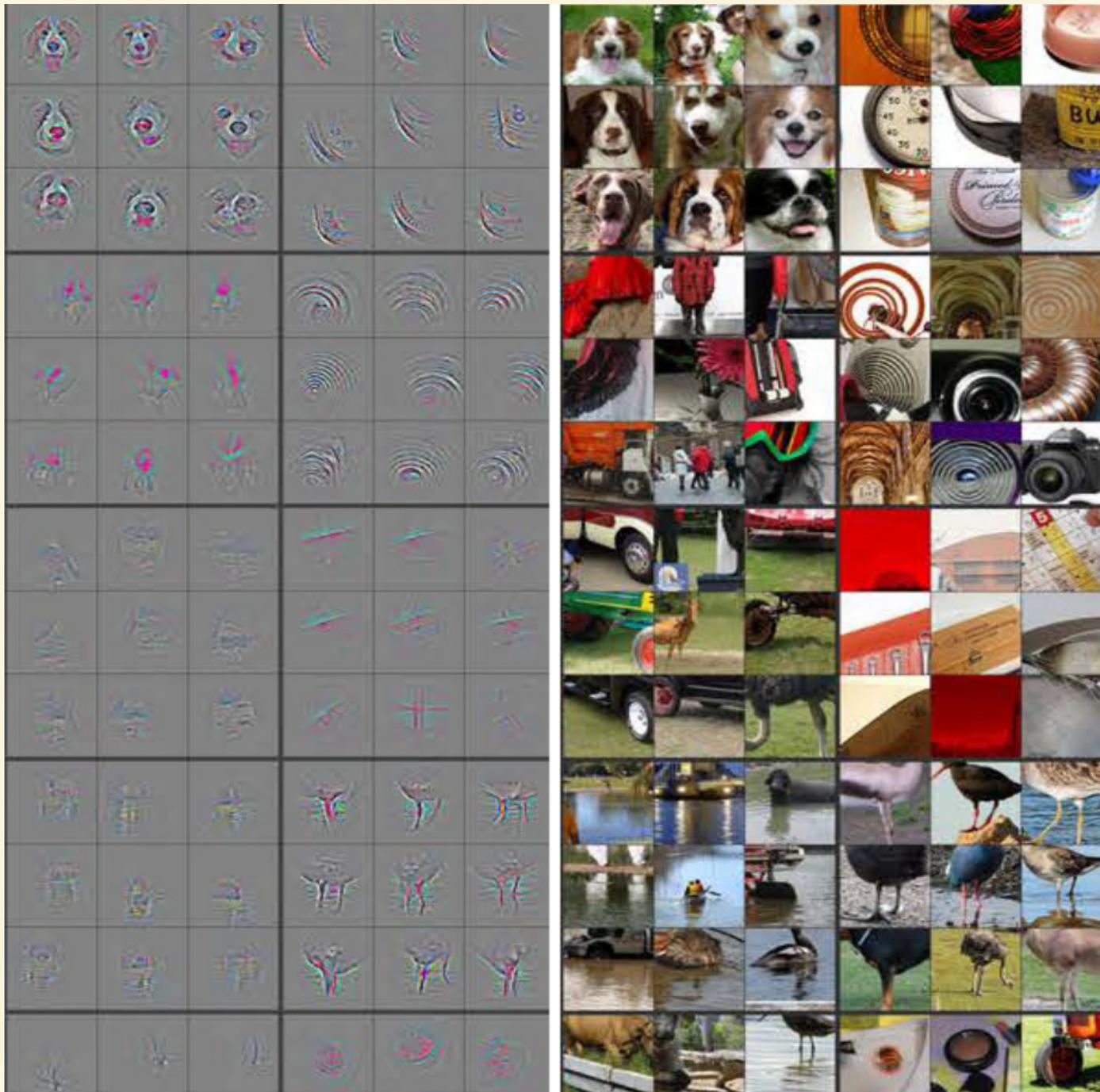
Layer 2





Layer 3





Reconocimiento de dígitos en línea

- [Link a colab](#)
- [Modelo en línea](#)