Universidad Tecnológica de Panamá

Facultad de Ingeniería de Sistemas Computacionales

Departamento de Computación y Simulación de Sistemas

Tópicos Especiales II - Código 0760: Machine Learning

Material Didáctico para la Asignatura

Preparado por:

José Carlos Rangel Ortiz, PhD.

2018

Índice general

Parte I

Machine Learning

Capítulo 1

Machine Learning

1.1. Historia

El machine learning es una sub rama de la inteligencia artificial , la cual a su vez es una rama de las ciencias de la computación . Comparte gran parte de su historia con el nacimiento de la inteligencia artificial, siendo en los últimos años una de las ramas con mayor auge de uso, investigación y desarrollo. El termino Inteligencia Artificial se acuña en la conferencia de Dartmouth de 1956. En esta se definen las directrices y líneas de actuación futuras. En 1958, Frank Rosenbalt diseña el perceptron, "la primera red neuronal artificial".

En los años 70 se aprecia el primer invierno de la IA debido a los cortes de financiación a las agencias de investigación en IA. Sin embargo el desarrollo de sistemas autónomos continuo su marcha a un ritmo lento. Durante este periodo (1967) se presenta el algoritmo de Vecinos Cercanos (Nearest Neighbor) el cual se considera el nacimiento del campo de reconocimiento de patrones. En los años 80s se vio el surgimiento de los sistemas expertos basados en reglas. Ante estos las empresas los adaptaron rápidamente, lo cual aumento el interés en el ML nuevamente.

El segundo Invierno de la IA llega a finales de los 80s e inicios de los 90s. A pesar de su corta duración, sus efectos fueron percibidos hasta entrados los 2000s. Durante este periodo se modifica el enfoque de trabajo del machine learning de un enfoque dirigido-por-conocimiento a un enfoque dirigido-por-datos. Se crean los primeros programas especializados en el análisis de una gran cantidad de datos. Un evento importante en el sector ocurrió en 1997 cuando el Computador Deep Blue de IBM vence en ajedrez al campeón mundial Gary Kaspárov.

Con la llegada del nuevo milenio, se aumento la potencia de calculo, al igual que la cantidad de datos disponibles para el análisis. Estos hechos han vuelto a lanzar el ML de tal manera que numerosas empresas están transformando sus negocios hacia los datos e incorporando nuevas técnicas de ML a sus procesos y servicios para obtener ventajas competitivas para su negocio.

En el 2006 el Término Deep Learning (Aprendizaje Profundo) es acuñado por Geoffrey Hinton el cual se usa para definir las nuevas arquitecturas de redes neuronales profundas, las cuales tienen mejores capacidades para extraer y aprender características de diversos tipos de datos.

Durante los últimos 10 años el ML ha tenido un fuerte auge a través del Deep Learning y las

diversas arquitecturas de redes neuronales. Se ha visto el surgimiento de proyectos como Google Brain, Deep Face(Facebook), se funda el OpenAI la cual busca que los desarrollos de la IA tengan un impacto positivo en la humanidad. Se han visto tambien es este periodo el lanzamiento de frameworks y kits de desarrollo de ML, creados por los gigantes de la tecnología como Microsoft, Amazon, Apple, Facebook, Tesla y Google. De igual manera el surgimiento de desafíos/competencias de clasificación como ImageNet y ModelNet han sido uno de los combustibles para continuar el crecimiento en el interés hacia el ML al igual de su uso en entornos cada día más cotidianos. En los últimos años el ML ha estado presente en la mayoría de las tecnologías que nos rodean desde un plancha de cocción eléctrica, frigoríficos, lavadoras, motores de redes sociales, servicios de streaming y coches de conducción autónoma entre otros.

En general la tendencia gobernante en la mayoría de los desarrollos de la IA, es el abandono de la busca de IA como una meta principal. En su lugar, se pretenden resolver problemas más concretos y que puedan ser fácilmente transferibles a la industria para que, poco a poco, los sistemas vayan adquiriendo inteligencia unido ello al desarrollo tecnológico y al progreso en campos afines [Miguel Cazorla, 2003].

1.2. Definición

> Machine Learning es la ciencia (y arte) de programar computadores de manera que estos puedan aprender a partir de los datos.
>
> Aurélien Géron

> **Definicion General**
> *Machine Learning* es el campo de estudio que da a los computadores la habilidad de aprender sin ser explícitamente programados.
>
> Arthur Samuel 1959.

> **Definicion orientada a Ingenieria**
> Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna tarea T y alguna medida de desempeño P, si su desempeño en T, al ser medido por P, mejora la experiencia E.
>
> Tom Mitchell,1997.

Un ejemplo clásico de Machine learning es el filtro de spam del correo electrónico. Este clasifica los correos según los ejemplos dados(etiquetados) por el usuario los cuales pueden ser: correos spam, o correo regular. Estos ejemplos que el sistema usa para aprender son llamados conjunto de entrenamiento (*training set*). Cada ejemplo es llamado una instancia de entrenamiento (*training instance*) o muestra.

En este caso la tarea T es etiquetar los nuevos correos, La experiencia E es el conjunto de entrenamiento y la medida de desempeño se debe definir, por ejemplo se puede utilizar la tasa de correos clasificados correctamente. Esta medida es llamada particularmente prescision (*accuracy*) y es usualmente la que se emplea en sistemas de clasificación.

1.3. Enfoques Tradicional y con *Machine Learning*

Porqué usar ML? [Géron, 2017]

Considerando el problema del *spam* usando técnicas de programación tradicional (Figura 1.1) el procedimiento era el siguiente:

1. Primero se debía mirar como luce un correo *spam* típicamente. Se notarían algunas palabras o frases (“Para ti”, “tarjeta de crédito”, “gratis” o “sorprendente”) las cuales tienen una tendencia mayor a aparecer en el asunto del mensaje. Quizás algunas de estas también en el nombre del remitente, el cuerpo del mensaje, etc.

2. Se procederá a escribir un algoritmo de detección para cada uno de los patrones que se han notado, y entonces el programa etiquetará los correos como *spam* si un numero de estas reglas son detectadas.

3. El programa se evaluara, y se repetirán los pasos 1 y 2 hasta que sea suficiente.

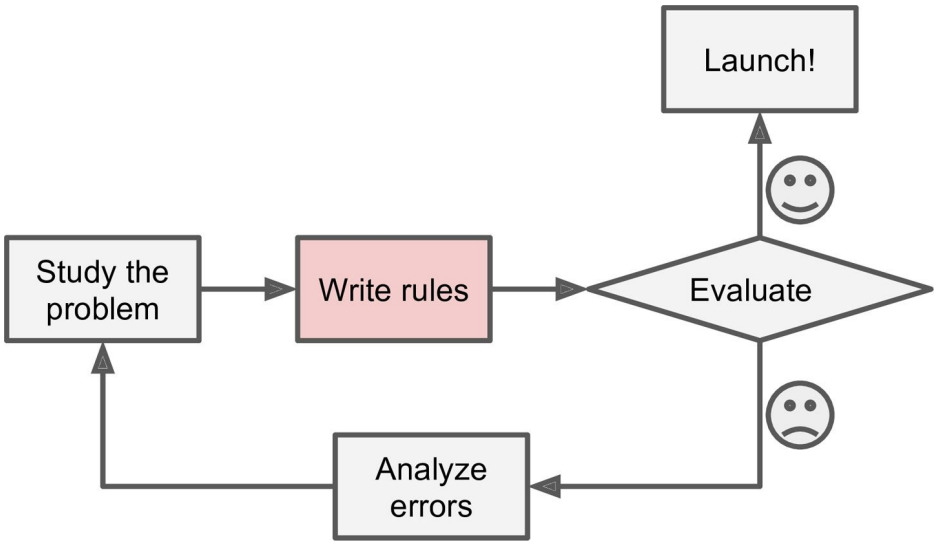


Figura 1.1: Enfoque Tradicional [Géron, 2017].

Debido a que este problema no es trivial, el programa sera probablemente en una larga y compleja lista de reglas, bastante difícil de mantener.

Al contrario que con un sistema basado en técnicas de ML (Figura 1.2) el cual aprende automáticamente que palabras y frases son buenas como predictoras de *spam* mediante la detección de patrones frecuentes inusuales, presentes en las muestras clasificadas como *spam* al compararlas con las muestras etiquetadas como correo ordinario. El programa seria mas corto, fácil de mantener y muy probablemente mas preciso.

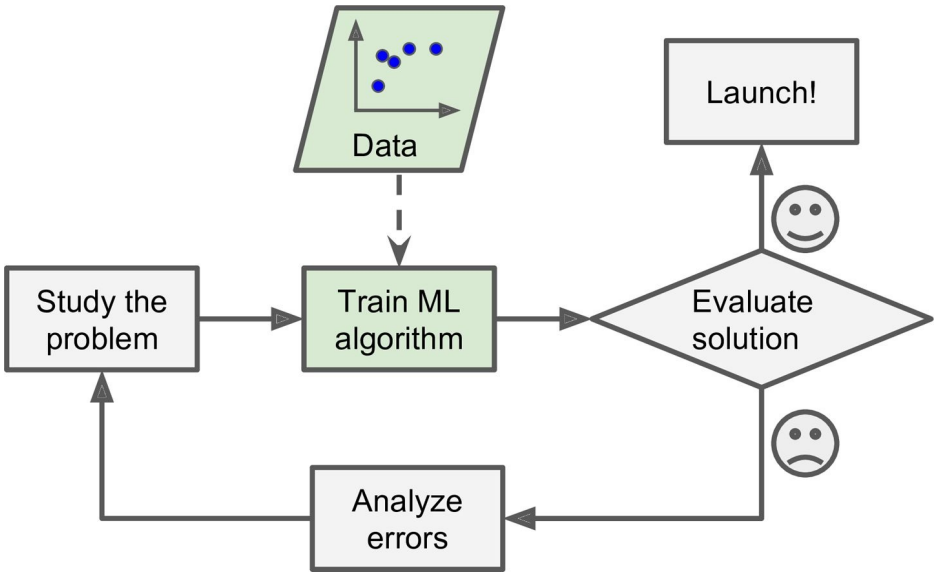


Figura 1.2: Enfoque aplicando ML [Géron, 2017].

Ademas, si los *spamers* notan que todos los correos que contienen la palabra "gratis" son bloqueados, estos pueden empezar a reemplazarla por la frase "sin costo alguno". Para un sistema tradicional, se deberían crear las reglas para detectar esta frase y así para cada nueva frase que se identifique como indicador de *spam*.

Al contrario que con un sistema basado en ML el cual automáticamente notaria que la frase "sin costo alguno" se vuelto in-usualmente frecuente en los correos identificados como *spam* por los usuarios.

Otro campo de uso de las técnicas de ML es el procesamiento de lenguaje, en este campo la complejidad del problema hace difícil el uso de técnicas tradicionales. Ya que debido a las diferencias de pronunciación que se aprecian en una palabra, debido a regionalismos y/o ruido ambiental entre otros. Con un sistema tradicional se tendrían que escribir reglas para cada uno de estos casos y para cada palabra. Por lo cual en la actualidad la mejor solución es escribir un algoritmo que aprenda automáticamente, dándole muchos ejemplos de palabras grabadas.

Los algoritmos de ML también pueden ayudar a los humanos a aprender. Estos pueden ser inspeccionados para ver que es lo que han aprendido (Figura 1.4). En el caso del spam podemos inspeccionar las palabras, frases o combinaciones, que ha aprendido a detectar como identificadores de correos *spam*. Lo cual en ocasiones puede llevar a un mejor entendimiento del problema al iden-

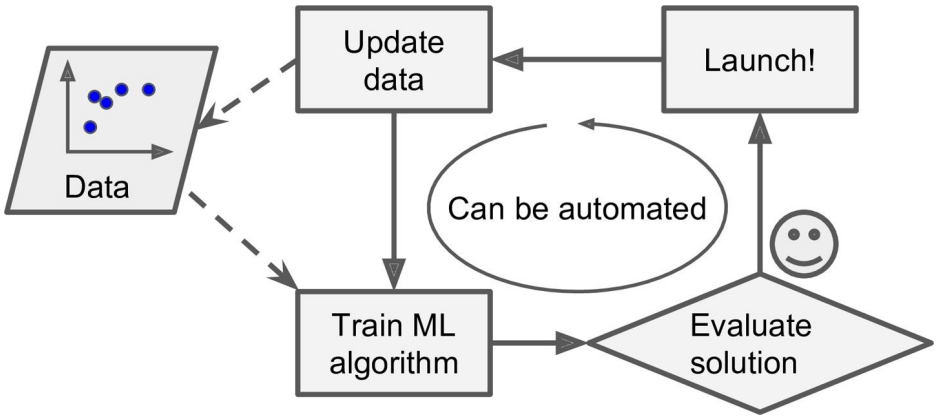


Figura 1.3: Enfoque con ML que se adapta automáticamente a los cambios [Géron, 2017].

tificar combinaciones no sospechadas o nuevas tendencias. Esta aplicación del ML para profundizar en grandes volúmenes de datos para ayudar a descubrir patrones se llama Minería de Datos (*Data Mining*).

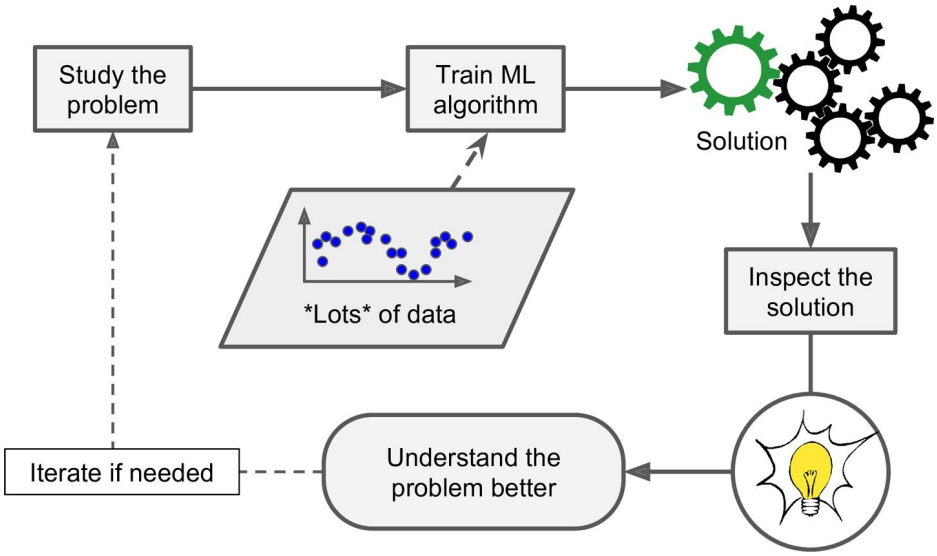


Figura 1.4: ML puede ayudar a los humanos a aprender[Géron, 2017].

Se puede resumir que el ML es bueno para:

- Problemas para los cuales las soluciones existentes requieren una gran cantidad de procesado manual o una larga lista de reglas: un algoritmo de ML, puede simplificar el código y desempeñarse mejor.
- Problemas complejos para los cuales no hay una buena solución con el enfoque tradicional: la mejor técnica de ML puede encontrar una solución.

- Ambientes fluctuantes: Los sistemas de ML se pueden adaptar a nuevos datos.
- Obtener información sobre problemas complejos y grandes volúmenes de datos.

1.4. Datasets

En el ML se habla constantemente de sistemas de clasificación o modelos de clasificación. Estos están construidos a partir del análisis de una gran cantidad de datos. Cuando una gran cantidad de datos se reúne, agrupa y organiza en categorías o secuencias, recibe el nombre de dataset. Son precisamente estos los que ha permitido aumentar el auge del machine learning en los últimos años. En la actualidad existen dataset para casi todo tipo de datos imágenes, panorámicas, nubes de puntos, objetos, señales de audio, señales eléctricas, imágenes medicas, videos, etc.

Los datasets generalmente se crean por parte de algún equipo de investigación, el cual se propone resolver un problema concreto y para ello requiere una gran cantidad de datos. Con el fin de que se publique y se pueda replicar su investigación, esta recopilación de datos se hace publica y accesible a otros grupos de investigación y/o empresas a los cuales les interesa resolver sino el mismo problema, uno similar para el cual los datos funcionan o se pueden adaptar.

Dependiendo de cada campo sus datasets populares variaran. Por ejemplo para reconocimiento de objetos están el ImageNet y el Pascal VOC, de igual manera para categorización de lugares se pueden encontrar Places y el New College.

Los dataset ademas de los datos incluyen información sobre como han sido capturados, guardados, etiquetados y procesados los datos, de igual manera cuales son los equipos empleados para la captura de datos. Con el fin de que el procedimiento seguido para generar un dataset pueda ser replicado para generar mas datos.

1.4.1. Descriptores

El ML se construye a base de procesos de entrenamiento, experimentación y validación de los modelos obtenidos. Para la construcción de dichos modelos utilizando los datasets seleccionados, la información de estos debe ser procesada en una manera general e uniforme para todo el conjunto. Generalmente un algoritmo de ML toma como entrada un conjunto de datos que ha sido representado a través de un descriptor el cual busca resaltar de manera general los atributos característicos que posee cada instancia de entrenamiento.

Los descriptores consisten en un vector multi-dimensional (Figura 1.5) en la cual cada dimensión describe el mismo atributo para todas las instancias del conjunto de datos. En otras palabras si la dimensión d_1 de un descriptor representa la cantidad de pixeles rojos de una imagen, entonces para cada muestra la dimensión d_1 representara este valor. De esta manera el algoritmo de ML tendrá como entrada el mismo tipo de información para cada muestra en el conjunto de entrenamiento.

	d_1	d_2	d_3	d_4	d_5	d_6	d_7	...	$l_{\|\mathcal{D}\|}$
	0.97	0.95	0.93	0.91	0	0	0	...	0
	0	0.94	0	0.92	0.93	0.94	0	...	0
	0.91	0	0	0	0.94	0.97	0.93	...	0
...	...	...	...	...	...	...	...	...	...
	0	0	0	0	0	0	0	...	0.91

Figura 1.5: Ejemplo de un descriptor

Los descriptores son generalmente arreglos numéricos de puntos flotantes, pero también descriptores que representan la información de manera binaria, representando la ausencia o presencia de un atributo en la muestra.

Los descriptores son generados utilizando algoritmos específicos los cuales se encargan de buscar características puntuales cuya descripción se almacenará en las diferentes dimensiones del descriptor. Estos algoritmos reciben también el nombre de descriptores y un nombre especifico el cual resume los atributos que busca el descriptor.

Como ejemplo de descriptores podemos mencionar SIFT, SURF para imágenes, Harris3D y Spin Images para nubes de puntos.

De igual manera también existen descriptores semánticos (Figura 1.6), en los cuales el valor de cada dimensión representa la probabilidad de representar la muestra con el concepto semántico de la dimensión.

	oficina	patio	baño	cocina	sala	ducha	portal	...	$l_{\|\mathcal{D}\|}$
	0.97	0.95	0.93	0.91	0	0	0	...	0
	0	0.94	0	0.92	0.93	0.94	0	...	0
	0.91	0	0	0	0.94	0.97	0.93	...	0
...	...	...	...	...	...	...	...	...	...
	0	0	0	0	0	0	0	...	0.91

Figura 1.6: Ejemplo de un descriptor semántico

Visualmente un descriptor se puede representar como un histograma donde cada columna representa una dimensión y la altura de la misma, el valor otorgado para ese atributo por el algoritmo.

Los algoritmos de descriptores se especializan en detectar esquinas, bordes para los casos de imágenes. Dependiendo del tipo de dato existen algoritmos específicos para su procesamiento y de igual manera un mismo tipo de dato puede ser procesado por diversos algoritmos de descriptores diferentes.

1.5. Clasificadores

Un clasificador de ML es un sistema que ha sido entrenado con un conjunto de datos los cuales le permiten al sistema aprender las características presentes en el conjunto de entrenamiento y a partir de ellas buscarlas en muestras nunca antes vistas con el objetivo de asignar una categoría a dicha información. En otras palabras un clasificador aprende a identificar la categoría de una muestra basándose en los atributos identificados en el conjunto de datos [Géron, 2017].

Los sistemas de ML se pueden dividir basándose en:

- Si estos son entrenados con o sin supervisión humana.
  - Supervisado (*Supervised*)
  - No-Supervisado (*Unsupervised*)
  - SemiSupervisado (*Semi-Supervised*)
  - Aprendizaje por Refuerzo (*Reinforcement Learning*)
- Si estos pueden o no aprender incrementalmente o en tiempo de ejecución
  - En Linea(*Online*)
  - Aprendizaje por Lotes (*batch learning*)
- Si estos trabajan simplemente comparando nuevos datos con los conocidos, o en su lugar detectan patrones en el conjunto de entrenamiento y construyen un modelo predictivo, similar a los científicos.
  - Aprendizaje basados en instancias (*instance-based learning*)
  - Aprendizaje basados en modelos (*model-based learning*)

Estos criterios no son exclusivos y se pueden combinar a la hora de construir un clasificador. Por ejemplo un clasificador de *spam* puede aprender en tiempo de ejecución usando una red neuronal entrenada usando muestras de correos *spam*/regulares, lo cual lo hace un sistema en linea, basado en modelos y supervisado.

1.5.1. Aprendizaje Supervisado/No-Supervisado

De acuerdo a la cantidad y el tipo de supervisión que recibe un sistema durante su entrenamiento este se puede clasificar en:

1.5.1.1. Aprendizaje Supervisado

En este tipo de aprendizaje, los datos que se usan para el entrenamiento incluyen la solución deseada para cada instancia, a la cual se le da el nombre de etiqueta. En otras palabras cada instancia tiene la información de la categoría a la cual pertenece (Figura 1.7).

En el ejemplo de detector de *spam*, el conjunto de entrenamiento contendrá las etiquetas de cada muestra (*spam* o ham), por lo tanto aprenderá a clasificar los nuevos correos.

Otro ejemplo típico consiste en predecir valor numérico especifico, por ejemplo el precio de un automóvil dado una serie de atributos (kilometraje, edad, marca, etc). Estos sistemas son llamados predictores y este tipo de tarea es llamada regresión. Para entrenar estos sistemas se les debe proveer de muchos ejemplos de automóviles incluyendo sus predicciones y sus etiquetas (precios en este caso).

Algunos de los sistemas supervisados más importantes son:

- k-*Nearest Neighbors* (k-Vecinos más cercanos)
- Regresion Lineal
- Regresion Logistica
- Maquinas de Vectores de Soporte (*Support Vectors Machines*-SVM)
- Arboles de Decisión y Bosques Aleatorios (*Decicion Trees and Random Forest*)
- Redes Neuronales

1.5.1.2. Aprendizaje no supervisado

En este tipo de sistemas el conjunto de entrenamiento no esta etiquetado. Es un sistema que trata de aprender sin un guiá/maestro (Figura 1.8).

Algunos de los tipos de sistemas no supervisados más importantes son:

- Agrupamiento(*Clustering*)

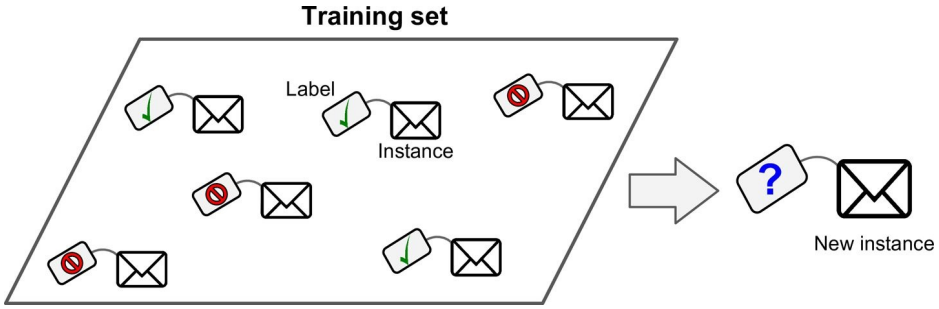


Figura 1.7: Un conjunto de entrenamiento etiquetado para utilizar en Aprendizaje Supervisado[Géron, 2017].

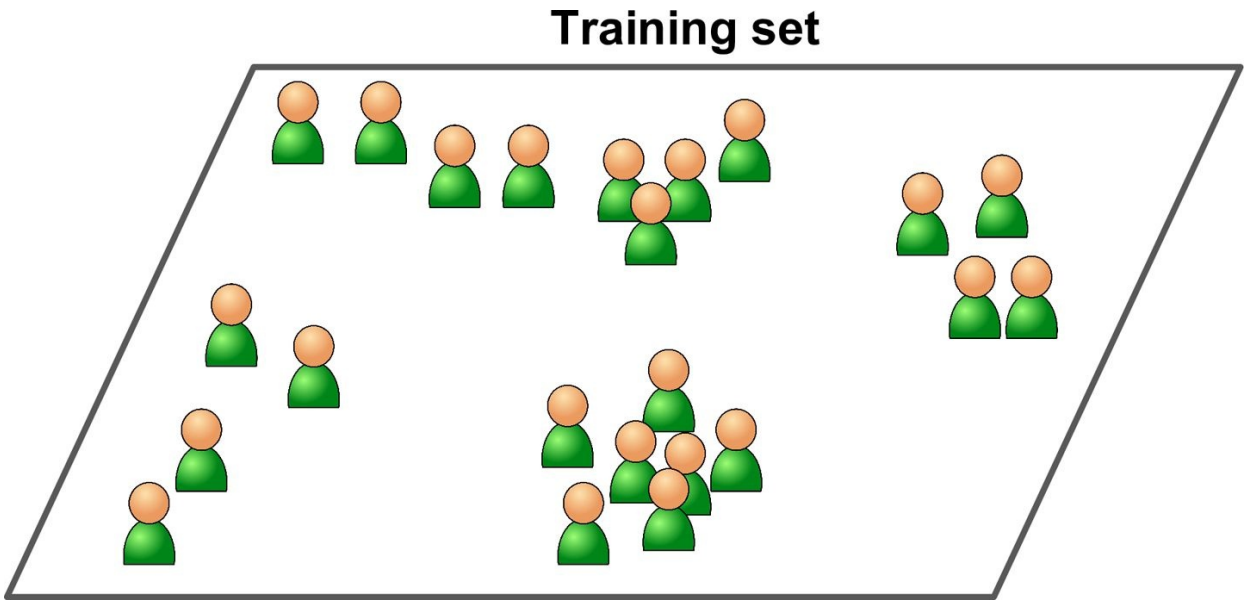


Figura 1.8: Un conjunto de entrenamiento no etiquetado para utilizar en Aprendizaje No Supervisado[Géron, 2017].

  - k-Means (K-medias)
  - Análisis de Agrupaciones Jerárquico
  - Maximizacion de la Expectativa
- Reducción y Visualización de la dimensionalidad
  - Análisis de componentes principales (PCA)
  - Kernel PCA
  - *Locally-Linear Embedding* (LLE)
  - *t-distributed Stochastic Neighbor Embedding* (t-SNE)
- Aprendizaje basado en asociación de reglas
  - Apriori
  - Eclat

Un ejemplo, si se tiene la información acerca de los visitantes de un blog. Se puede ejecutar un algoritmo de agrupamiento/clustering que tratara de detectar grupos de visitantes similares. En ningún momento se le indica al algoritmo a que grupo pertenecen los visitantes, el algoritmo en si encuentra esas conexiones sin ayuda. Como resultado estos pueden notar que el 40 % de los visitantes son hombres a los cuales les gustan los comics y que generalmente leen el blog en las tardes, mientras el 20 % son jóvenes amantes de la ciencia ficción los cuales visitan el blog en los

fines de semana. Si se usa un algoritmo de **clustering** jerárquico se puede incluso dividir cada grupo en sub grupos más pequeños, lo cual pueden ayudar al redactor del blog a crear post para cada grupo (Figura 1.9).

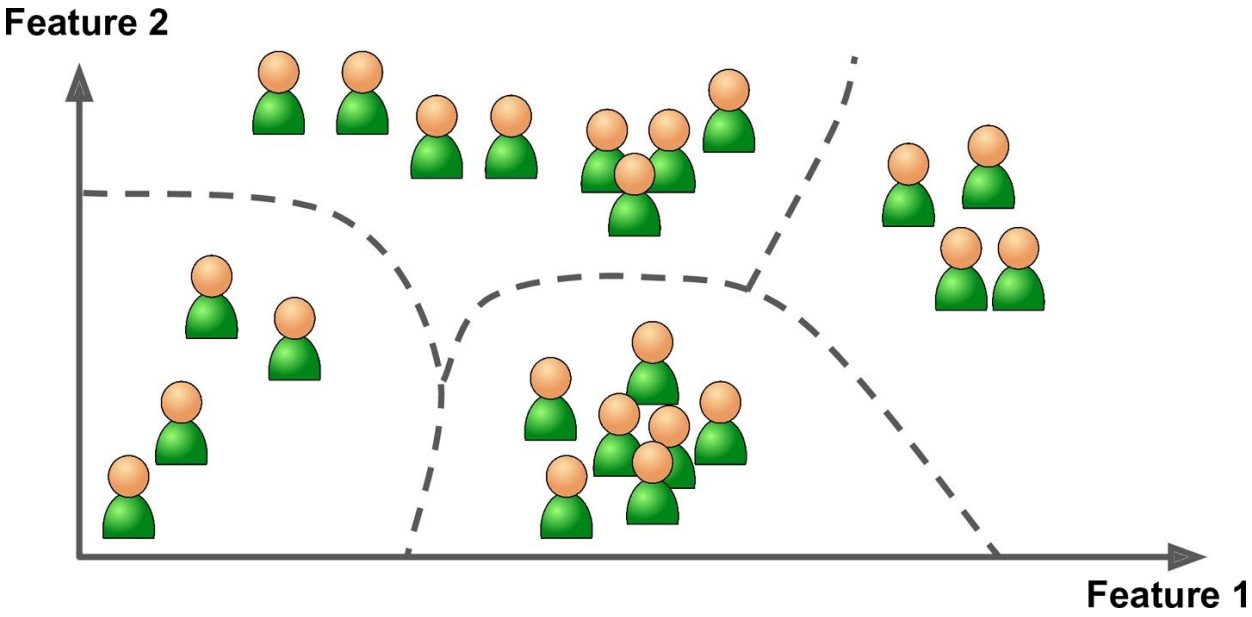


Figura 1.9: Agrupamiento[Géron, 2017].

Los algoritmos de visualización son ejemplos algoritmos no supervisados. Reciben una gran cantidad de datos complejos y no etiquetados, la salida es una representación 2D o 3D de los datos la cual puede ser fácilmente graficada (Figura 1.10)

Un ejemplo de uso de estos sistemas es para la detección de anomalías de funcionamiento (Figura 1.11.

1.5.1.3. Aprendizaje semisupervisado

Estos algoritmos son una combinación de los algoritmos supervisados y no supervisados. Estos son entrenados secuencialmente en una manera no supervisada y el sistema completo es ajustado usando técnicas de aprendizaje supervisado (Figura 1.12).

1.5.1.4. Aprendizaje por Refuerzo

Este tipo de aprendizaje es muy diferente a los demás. En este un sistema llamado agente puede observar el entorno, seleccionar y ejecutar acciones, entonces podrá obtener recompensas (en caso de éxito) y penalizaciones (en caso de fallo), como se muestra en la Figura 1.13. Este aprenderá por si mismo cual es la mejor estrategia, llamada política, para obtener recompensas la mayor parte del tiempo. Esta política define la la acción que ejecutara el agente cuando se presenta una determinada situación

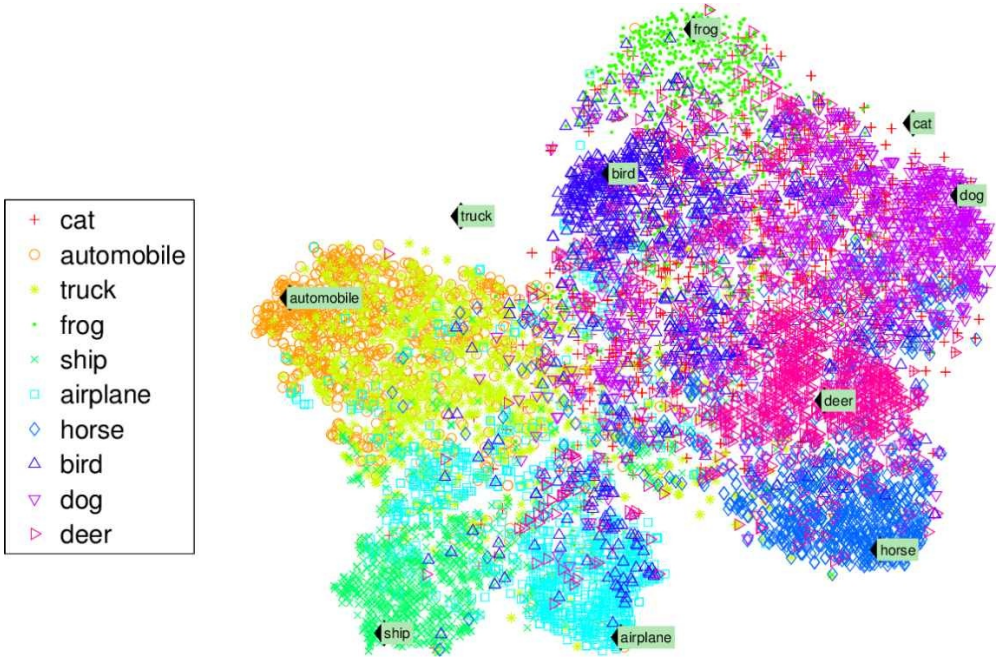


Figura 1.10: Ejemplo de visualización de *clusters* generados por un algoritmo de agrupamiento[Géron, 2017].

Figura 1.11: Detección de anomalías[Géron, 2017].

1.5.2. Aprendizaje en linea y por por lotes

Otro criterio para clasificar sistemas de ML es si el sistema puede aprender o no incrementalmente desde un flujo de datos entrante.

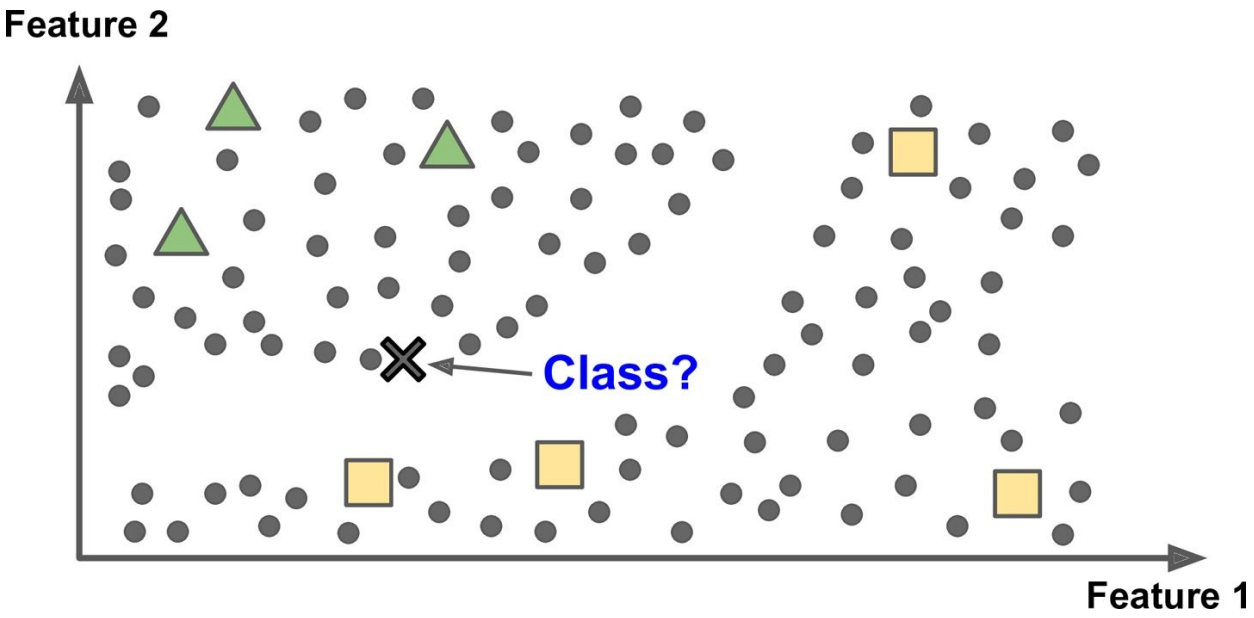


Figura 1.12: Aprendizaje Semi-Supervisado [Géron, 2017].

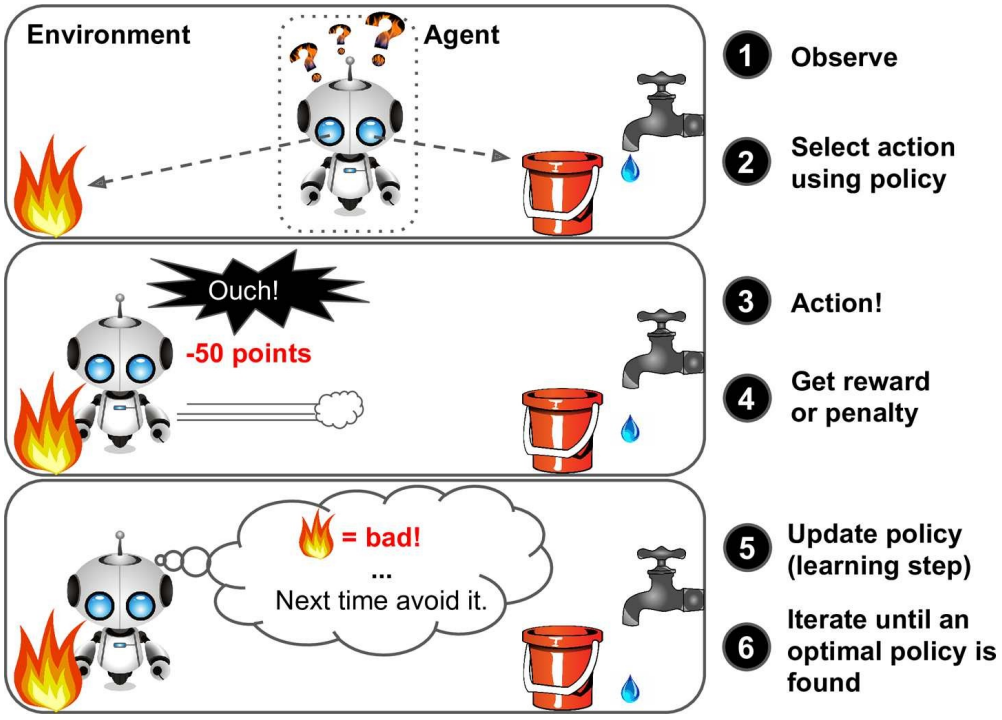


Figura 1.13: Aprendizaje por refuerzo [Géron, 2017].

1.5.2.1. Aprendizaje por lotes

En el aprendizaje por lotes, el sistema es incapaz de aprender incrementalmente: este debe ser entrenado utilizando todo el conjunto disponible de datos. Esto toma generalmente mucho tiempo

y recursos computacionales, por lo tanto esto es hecho *offline*. Primero el sistema es entrenado y entonces es ejecutado sin aprender más nunca; este simplemente aplica lo que ha aprendido. Esto es llamado aprendizaje *offline* o fuera de linea.

Si se desea que el sistema conozca nuevos datos, es necesario entrenar un a nueva versión del sistema desde 0 con el conjunto completo de datos (no solo con la nueva información, sino también con todos los datos), luego se debe detener el sistema antiguo y remplazarlo con el nuevo.

Afortunadamente el proceso completo de entrenar, evaluar y lanzar un sistema de ML puede ser automatizado de manera fácil, como se aprecia en la Figura 1.3, incluso un sistema con aprendizaje por lotes se puede adaptar al cambio. Simplemente actualizando los datos realizando un entrenamiento desde 0 tan a menudo como se necesite.

Esta solución es simple y a menudo funciona bastante bien, pero entrenar todo el sistema en el conjunto completo de datos puede tomar muchas horas, por lo cual generalmente este entrenamiento se programa para realizarse cada 24 horas o inclusive de manera semanal. Si el sistema requiere aprender constantemente de un flujo de datos rápido y cambiante quizás se necesite una solución más reactiva.

El entrenamiento con el conjunto completo de datos requiere de mucho recursos computacionales (procesador, espacio en memoria, espacio en disco, I/O a disco y a red, etc.). Si se tienen muchos datos y se programa al sistema para re-entrenar constantemente, esto terminará constando una enorme suma de dinero. Si la cantidad de datos es inmensa, entonces puede ser imposible utilizar un algoritmo de aprendizaje por lotes.

Si el sistema necesita aprender automáticamente y tiene recursos limitados (aplicación de *smartphone* o un *rover* en Marte), entonces cargar con una gran cantidad de datos y entrenar cada día por unas horas es contraproducente.

Afortunadamente, una mejor opción en todos estos casos es emplear algoritmos que son capaces de aprender de manera incremental.

1.5.2.2. Aprendizaje en Linea

En el aprendizaje en linea se puede entrenar el sistema de manera incremental mediante la adición secuencial de instancias de datos o mediante el uso de pequeños grupos llamados *mini-batches*. Cada paso de aprendizaje es rápido y barato, por lo tanto el sistema puede aprender acerca de los nuevos datos sobre la marcha, a medida que estos llegan (Figura 1.15).

El aprendizaje en linea es bueno para sistemas que reciben información como un flujo continuo y los cuales deben adaptarse al cambio de manera rápida o autónoma , por ejemplo el precio de las acciones. Es también una buena opción si se cuenta con recursos computacionales limitados: una vez se que un sistema de aprendizaje en linea ha aprendido sobre nuevas instancias de datos, este no las necesita más, por lo cual se pueden descartar, lo cual puede ahorrar una gran cantidad de espacio de almacenamiento.

El algoritmo de aprendizaje en linea se pueden emplear para entrenar sistemas que requieren

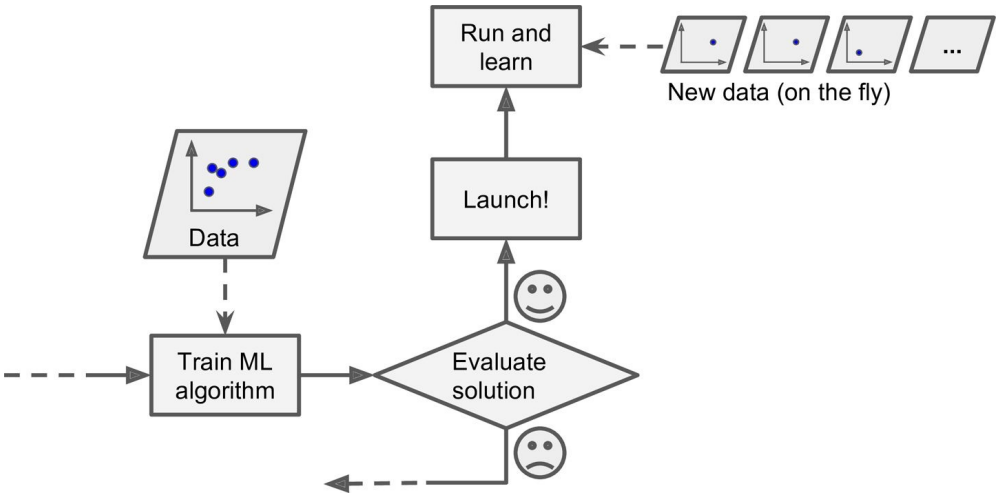


Figura 1.14: Aprendizaje en linea[Géron, 2017].

un dataset gigantesco, el cual no se puede ajustar en la memoria principal de la maquina (a esto se le conoce como aprendizaje fuera del núcleo). El algoritmo carga parte de los datos, ejecuta un paso de entrenamiento en esos datos y repite el proceso hasta que se halla utilizado todo el conjunto de datos (Figura **??**).

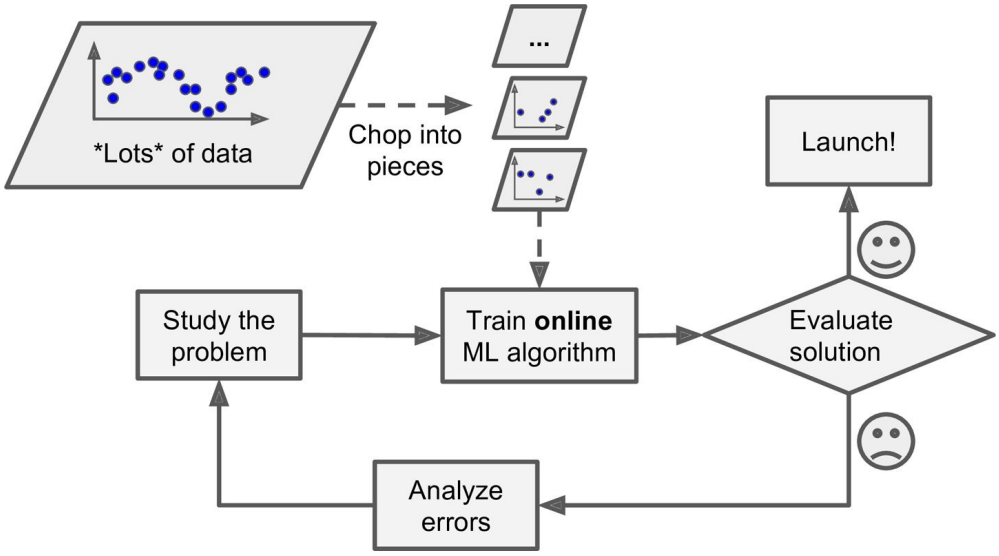


Figura 1.15: Uso de aprendizaje en linea para manejar datasets grandes[Géron, 2017].

Un parámetro importante en estos sistemas es como de rápido estos se deben adaptar a los datos cambiantes: esto es llamado tasa de aprendizaje (lr) (*learning rate*) . Si se tiene un lr alto, entonces el sistema se adaptara rápidamente a nuevos datos, pero tenderá a olvidar los datos antiguos mas rápidamente. Por el contrario, si se usa un lr bajo el sistema aprenderá más lentamente y también será menos sensitivo al ruido en los datos nuevos o a secuencias de datos no representativos.

Un gran reto con el aprendizaje en linea es que si se introducen malos datos en el sistema el desempeño del sistema caerá gradualmente. Si se trata de un sistema en vivo, los cliente lo notarán. Para reducir este riesgo, el sistema se debe supervisar de manera cercana y en cuanto se detecte una caída en el desempeño, realizar un cambio en el lr. Es también buena idea utilizar un algoritmo de detección de anomalías, para supervisar los datos de entrada.

1.5.3. Aprendizaje Basado-en-Modelos y Aprendizaje Basado-en-Instancias

Otra división en los sistemas de ML se basa en como generalizan. La mayoría de las tareas de los sistemas de ML son acerca de realizar predicciones. Lo que significa que dado un numero de instancias de entrenamiento, el sistema necesita ser capaz de generalizar para instancias que no ha visto anteriormente. Debido a que un buen desempeño sobre las instancias de entrenamiento es bueno, pero no suficiente, debido a que el objetivo principal es ejecutare en nuevas instancias.

1.5.3.1. Aprendizaje Basado-en-Instancias

Es posiblemente una de las formas más fáciles de aprendizaje. En el caso del detector de filtro , el sistema etiquetaría todos los correos que fueran idénticos a los que han sido etiquetados por los usuarios, no es la peor solución pero tampoco la mejor.

Como otro enfoque se puede clasificar como *spam* los correos que son ligeramente similares a los etiquetados por el usuario, para lo cual se requiere de una medida de similitud entre dos correos. Por ejemplo una medida podría ser la cantidad de palabras en las que coinciden. Por lo cual el sistema clasificara el correo como *spam* si existen muchas palabras en común con una instancia de entrenamiento.

Por lo tanto en esto sistemas se aprende los ejemplos de memoria, y entonces generaliza a nuevos casos usando una medida de similitud (Figura 1.16).

1.5.3.2. Aprendizaje Basado-en-Modelos

Otra manera de generalizar a partir de un conjunto de ejemplos es la construcción de un modelo de estos ejemplos, luego se emplea este modelo para hacer las predicciones (Figura 1.17).

Un ejemplo de tipo de sistema se puede construir si se desea saber si el dinero hace a las personas felices, para crear el conjunto de entrenamiento se pueden descargar los datos del *Better Life Index* del sitio web de OECD, así como también las estadísticas GDP per capita. Luego uniendo estas se puede crear una tabla como se observa en la Tabla 1.1. La gráfica de estos datos se muestra en 1.18

En esta gráfica se puede observar una tendencia, a pesar de la presencia de cierta cantidad de ruido en los datos. Se puede observar la tendencia de que que la satisfacción aumenta conforme aumenta el GDP , por lo cual se decide modelar la satisfacción de como función del GPD. A este

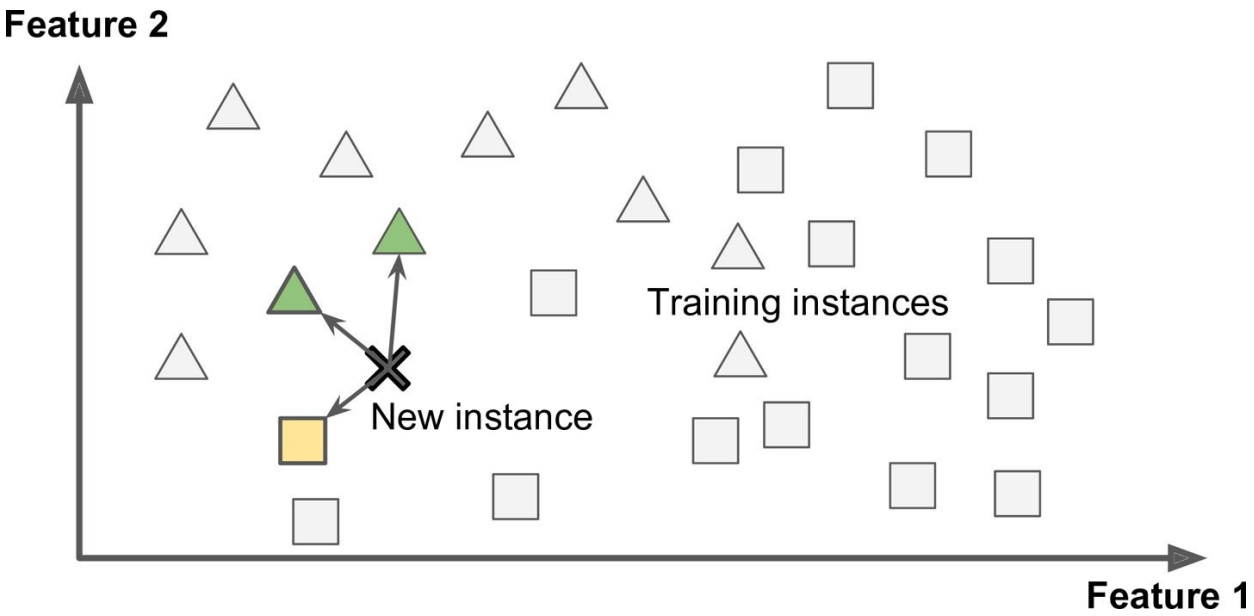


Figura 1.16: Aprendizaje Basado en Instancias[Géron, 2017].

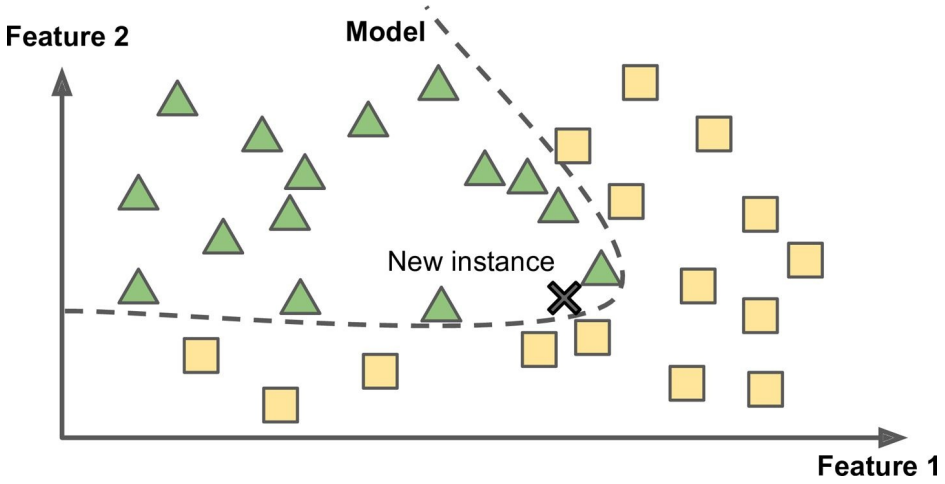


Figura 1.17: Aprendizaje Basado-en-Modelos[Géron, 2017].

Cuadro 1.1: Tabla de ejemplo para el problema.

País	GDP	Satisfacción
Hungría	12,24	4,9
Corea	27,195	5,8
Francia	37,675	6,5
Australia	50,962	7,3
EEUU	55,805	7,2

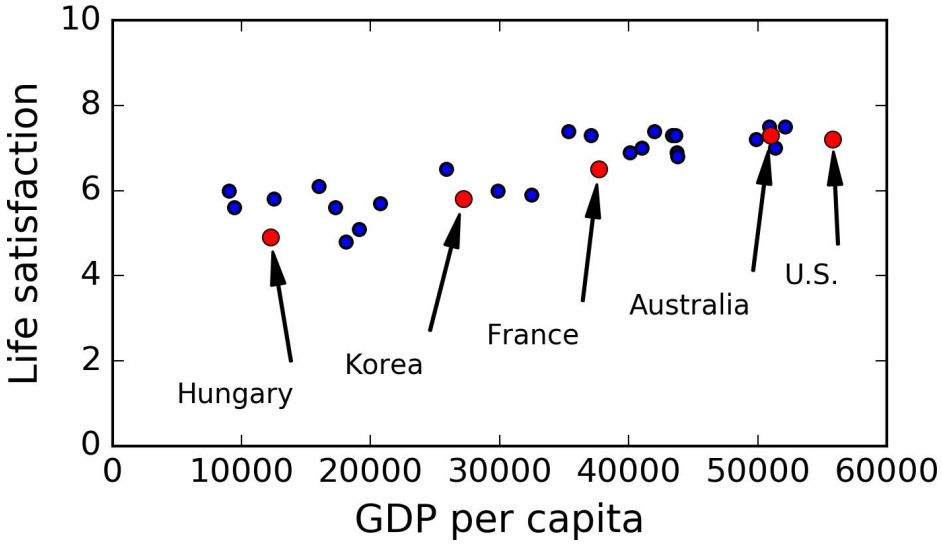


Figura 1.18: GDP per capita para los países en el ejemplo[Géron, 2017].

paso se le llama selección del modelo, debido a que se selecciona un modelo lineal con solo un atributo.

$lifeSatisfaction = \theta_0 + \theta_1 \times GDP\_per\_capita$

Este modelo tiene dos parámetros, mediante la selección de estos parámetros, se puede hacer que el modelo se pueda representar con cualquier función lineal, como se ve en la Figura 1.19.

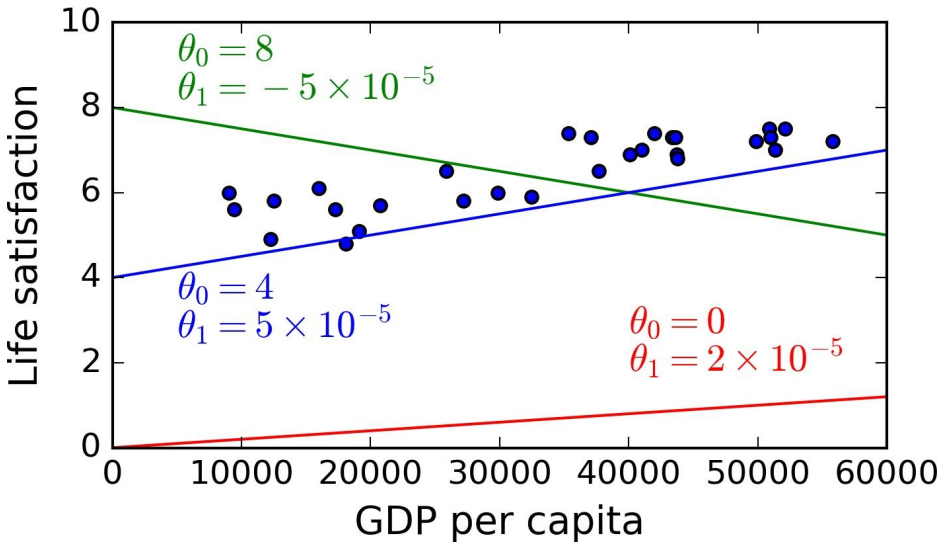


Figura 1.19: Posibles hipótesis para la construcción del modelo[Géron, 2017].

Antes de utilizar el modelo, se deben definir los parámetros $\theta_0 y \theta_1$. Para conocer cuales son los parámetros permitirán el mejor desempeño al modelo, se debe especificar una medida desempeño.

Para ello se puede definir una función de ajuste la cual mide como de bueno es el modelo, o se puede definir una función de costo que mide como de malo es el modelo. Para la regresión lineal se suele definir una función de costo, la cual mide la distancia entre la predicción hecha por el modelo y el valor real de la instancia de entrenamiento, el objetivo es disminuir el valor de dicha distancia.

En este punto es donde el algoritmo de regresión lineal es utilizado: se debe proveer al sistema con los ejemplos de entrenamiento y este encontrará los parámetros que se ajusten mejor a los datos de entrenamiento. Esto es conocido como entrenamiento del modelo, para ese caso el algoritmo encontró que los mejores parámetros son $\theta_0 = 4{,}85$ y $\theta_1 = 4{,}91 \times 10^{\breve{}5}$.

Ahora el modelo se ajusta a los datos lo más cerca posible para un modelo lineal, como se aprecia en el figura 1.20.

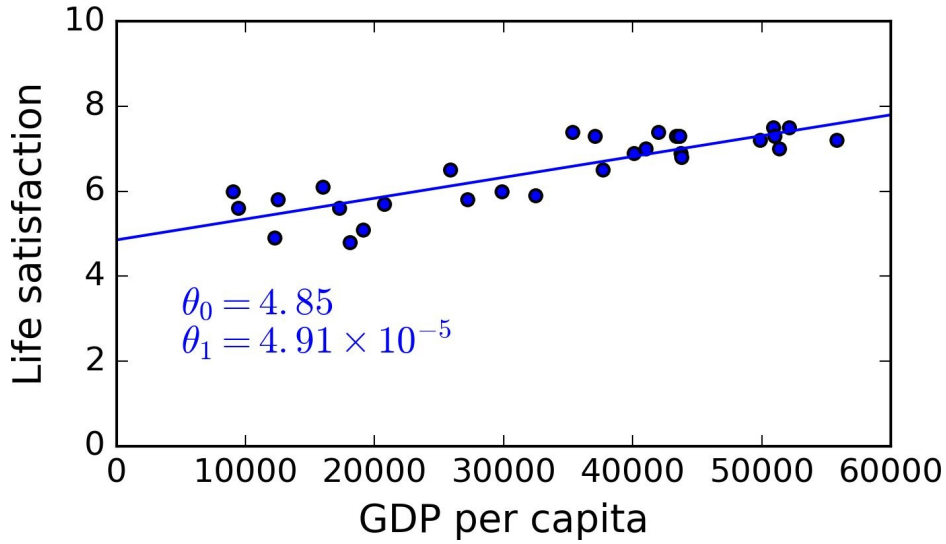


Figura 1.20: Parámetros Seleccionados para la construcción del modelo[Géron, 2017].

Finalmente el modelo esta listo para ser ejecutado y producir predicciones. Por ejemplo si se desea saber que tan felices son los Chipriotas y no se tiene ningún registro en la OECD, se puede utilizar el modelo para hacer una predicción empleando el GDP per capita de Chipre.

1.5.4. Técnicas de construcción

1.5.4.1. *Boosting*

Se refiere a cualquier método que puede combinar varios clasificadores débiles en un clasificador más fuerte o robustos. La idea general de la mayoría de los métodos de *boosting* es entrenar predictores secuencialmente, de manera que cada uno trata de corregir su antecesor. Existen muchos métodos de *boosting*, pero uno de los más popular es *AdaBoost* (*Adaptative Boosting*).

1.5.4.2. *AdaBoost*

Una manera de para un nuevo predictor de corregir su predecesor es prestar más atención a las instancias de entrenamiento a las cuales su predecesor no logro ajustarse. Esto resulta en nuevos predictores que se enfocan más en los casos difíciles. En esto consiste *AdaBoost*.

Por ejemplo para construir un clasificador con *AdaBoost* el primer clasificador se entrena para hacer predicciones en un conjunto de entrenamiento. El peso relativo de las instancias mal clasificadas es entonces incrementado. Un segundo clasificador se entrena usando los pesos actualizados y nuevamente este hace predicciones en el conjunto de entrenamiento. Por lo cual los pesos son actualizados sucesivamente (Figura 1.21).

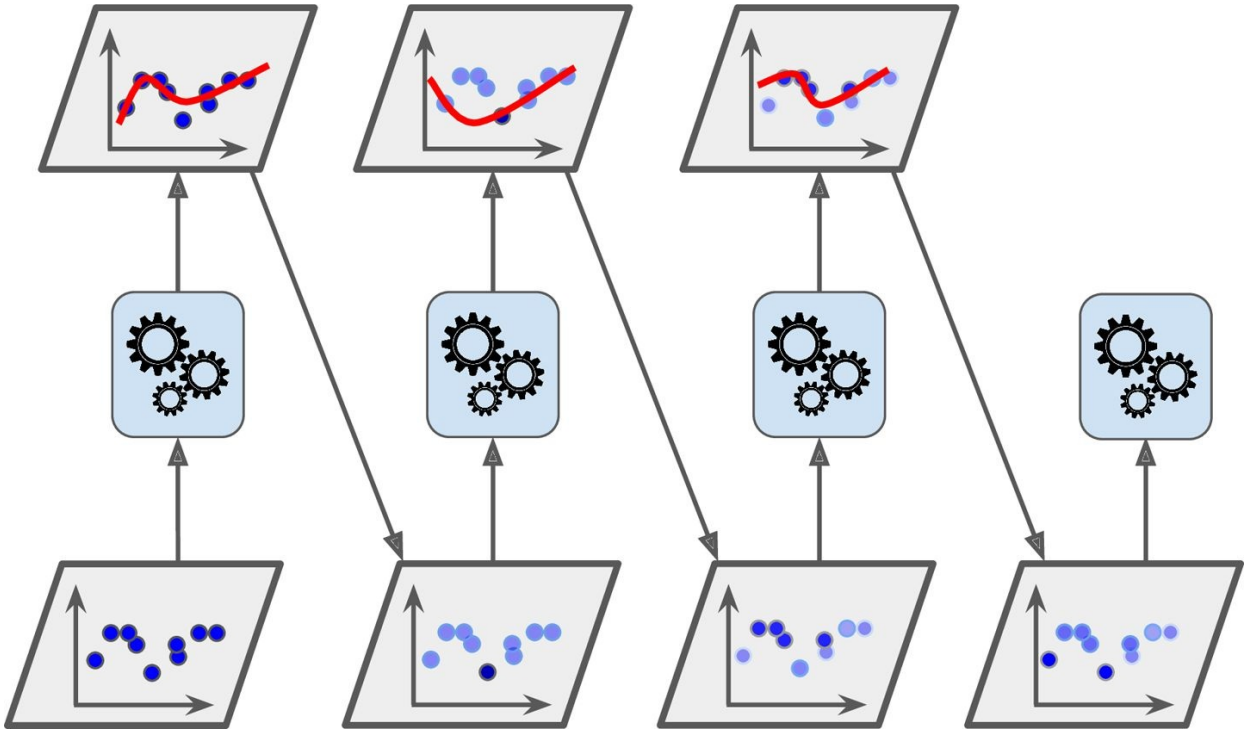


Figura 1.21: Entrenamiento secuencial con *AdaBoost*, se aprecia la actualización de los pesos de las instancias[Géron, 2017].

1.5.4.3. *Voting*

Podemos suponer que hemos entrenado algunos clasificadores (Figura 1.22), de los cuales cada uno obtiene un 80 % de precisión. Se pueden tener por ejemplo un clasificador de regresión logística, una maquina de vectores de soporte (SVM), un clasificador de bosques aleatorios, un clasificador k-NN y algunos cuantos más.

Una manera sencilla de crear un mejor clasificador es agrupar todas las predicciones de cada clasificador y el resultado final sera la predicción con más votos. Se conoce también como un

clasificador de mayoría de votos (Figura 1.23).

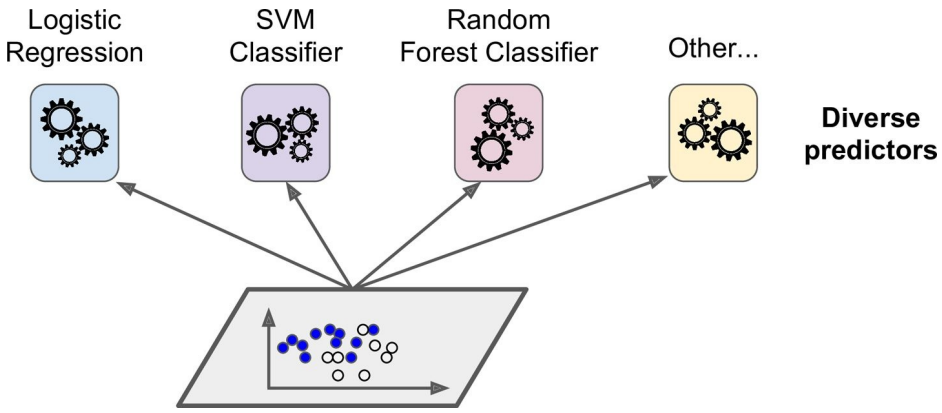


Figura 1.22: Entrenamiento de diversos clasificadores[Géron, 2017].

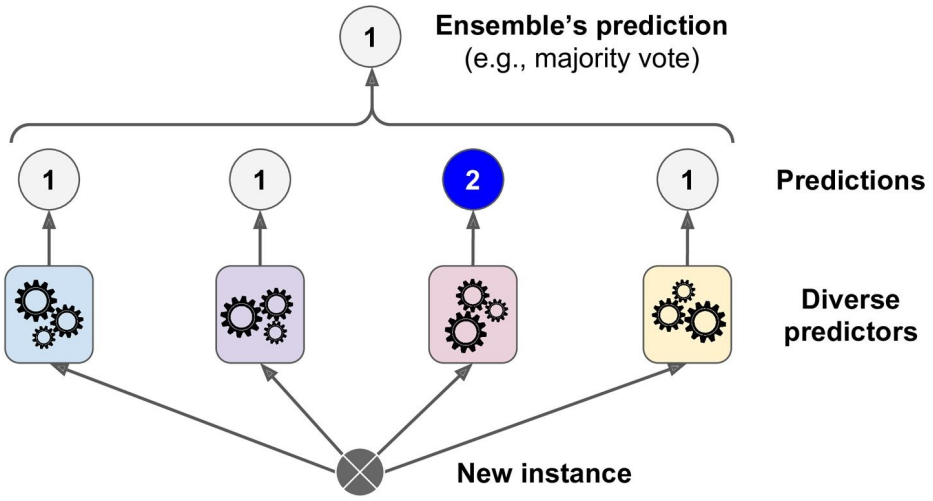


Figura 1.23: Prediccion producida por el procedimiento mediante *Voting*[Géron, 2017]

1.6. Clasificadores Binarios/Multiclase

Esta división de clasificadores se enfoca en la cantidad de predicciones que puede generar un clasificador.

1.6.1. Clasificadores Binarios

Un clasificador binario puede predecir si una instancia nunca antes vista pertenece o no a una categoría, por ejemplo el detector de *spam* del correo electrónico. Es un clasificador del tipo positivo/negativo para las muestras del estudio.

1.6.2. Clasificadores Multiclase

Un clasificador multiclase por el contrario es capaz de aprender a identificar entre varias categorías, por lo tanto las instancias de entrenamiento provienen de varios tipos de objetos. Un ejemplo común es la clasificación de lugares, donde un clasificador puede detectar si se esta en una oficina, salón de clases o laboratorio.

1.7. Evaluación de Clasificadores

Una vez se ha construido un clasificador, es importante conocer su desempeño en la tarea para cual fue diseñado. Este medición se hace generalmente calculando como de preciso es el clasificador e inclusive tratar de visualizar las categorías que pueden representar un problema para el sistema.

1.7.1. Precisión del Clasificador

El cálculo de la precisión del clasificador se hace mediante la siguiente ecuación:

$$precisión = \frac{numero\_de\_aciertos}{numero\_total\_de\_instancias\_en\_el\_dataset}$$

Esta indica cuantas instancias fueron identificadas adecuadamente por el sistema. Esta medida se emplea también para sistemas de clasificación multiclase.

1.7.2. Matriz de Confusión

Una mejor manera de evaluar el desempeño de un algoritmo es mirar la matriz de confusión. La idea general tras esta, es contar el numero de instancias de la clase A fueron erroneamente clasificados como la clase B. Por ejemplo se desea saber el numero de veces que el clasificador confundió una imagen de la categoría 5 (Figura 1.24). La diagonal principal de dicha matriz indica la precision del clasificador en cada categoría que intervino en el entrenamiento.

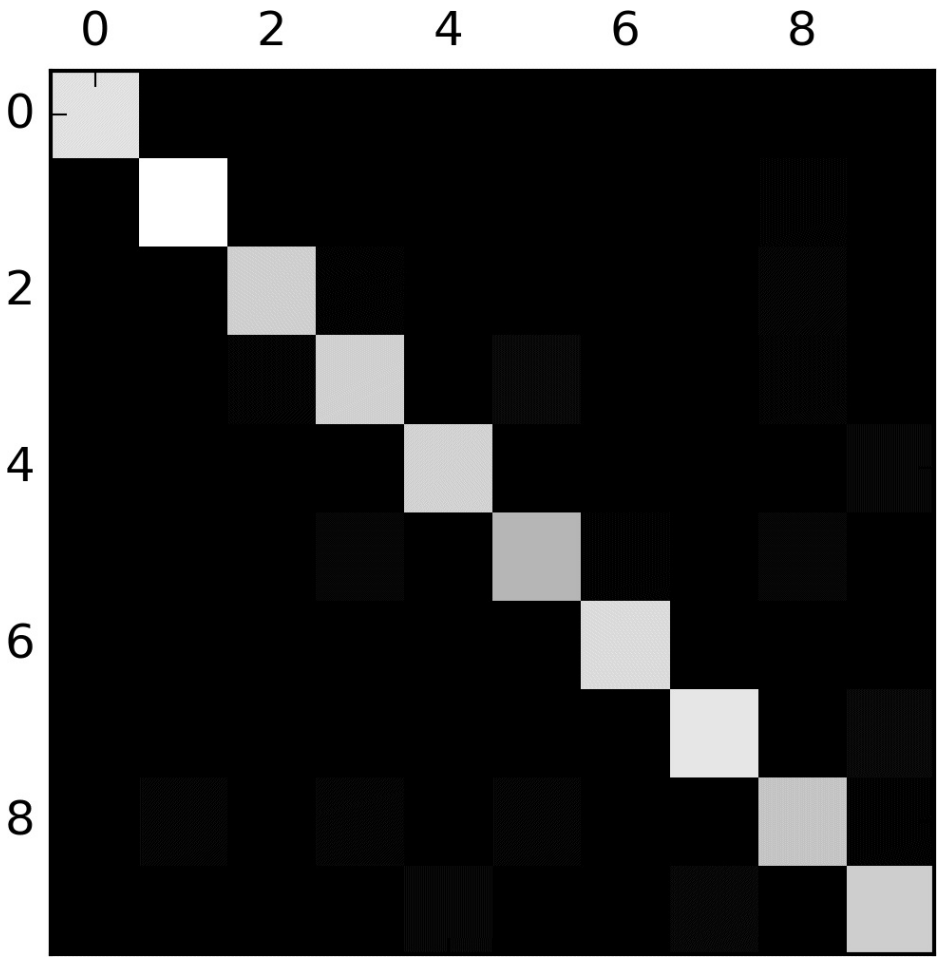


Figura 1.24: Matriz de confusión para un clasificador de numeros.

Capítulo 2

Metodos de Construcción de Clasificadores

2.1. Herramientas para Desarrollo

Para construir un clasificador uno de los primeros pasos es la selección del algoritmo que se utilizará para el mismo. Cuando se tenga esa elección, se debe entonces buscar la implementación de dicho algoritmo y por ende la herramienta con la que se construirá. Existen muchas herramientas en el mercado para construir clasificadores, por lo tanto es muy común que para un mismo algoritmo se cuente con varias opciones para su desarrollo en distintos lenguajes e incluso versiones para ejecución en procesadores gráficos. Las herramientas para desarrollo pueden ser desde una librería en un lenguaje especifico, una aplicación completa con los diferentes algoritmos o incluso sistemas de servidores en la nube. En esta sección se hablará y utilizaran el entorno Weka para Java y el conjunto de librerías SciKit-Learn para Python. Como librerías especificas se mencionara LibSVM la cual se puede utilizar en diversos lenguajes de programación.

2.1.1. Weka

Figura 2.1: Logo del proyecto Weka.

Weka es un software de minería de datos escrito en Java, el cual integra la implementación de varios algoritmos de análisis y procesamiento de datos. Estos algoritmos pueden ser aplicados directamente a un dataset o llamados desde nuestro propio código fuente en Java. Weka contiene herramientas para el pre-procesado y análisis de datos, modelado predictivo clasificación, regresión, agrupamiento, reglas de asociación, también tiene herramientas para la visualización de estos datos, además provee una interfaz gráfica que unifica las herramientas para que estén a una mejor disposición. Es incluso adecuado para el desarrollo de nuevos esquemas de ML.

Es desarrollado por la Universidad de Waikato en Nueva Zelanda desde 1993. El nombre proviene de un ave que no vuela endémica de las islas de Nueva Zelanda, aunque comúnmente se ha definido también como Waikato Environment for Knowledge Analysis además es una herramienta de distribución de licencia GNU-GLP o software libre.

Todas las técnicas en WEKA están basadas en la asunción de que los datos están disponibles en un fichero de texto plano o una relación, en donde cada registro de datos esta descrito por un número fijo de atributos nominales o numéricos.

Permite el acceso a otras instancias de bases de datos por medio de SQL, gracias al JDBC, además puede procesar un resultado generado a base de una consulta hecha a una base de datos.

2.1.1.1. Entornos disponibles en Weka

Para ejecutar Weka desde su ejecutable se escribe lo siguiente en una consola de comandos:

```
$ java -Xmx2G -jar weka.jar
```

El segundo parámetro -Xms2G indica cuanta memoria adicionales le asigna para la ejecución del sistema. En este comando el $2G$, significa que se han asignado 2 *gigabytes* más para la ejecutable del sistema.

Una vez ejecutemos Weka, se nos preguntará que tipo de entorno deseamos utilizar, esto dependerá de lo que deseamos hacer (Figura 2.2). De igual manera es posible utilizar Weka solamente desde la consola de comandos (Command Line Interface)

1. **Simple CLI** :básicamente se trata de una consola la cual permite tener acceso a todas las funciones pero sujeta a la línea de comandos, todas son invocadas por medio de Java, puede realizar cualquier operación soportada por weka pero de manera directa.

2. **Experimenter**: El modo experimentador muy útil para aplicar uno o varios tipos experimentos pero a gran escala, por ejemplo se puede hacer clasificación sobre un gran conjunto grande de datos, para luego proceder a realizar análisis estadístico y por ultimo obtener índices de estadística.

3. **Explorer**: Se puede decir que es la interfaz de usuario que permite más accesibilidad a los componentes principales de todo el bando de trabajo de modo gráfico, contiene el acceso a todas las funciones de manera muy sencilla, este está dividido en varios paneles (Figura 2.3).

   - *Preprocess*: Aquí están incluidas todas las herramientas y filtros para cargar y manipular los datos que se pretende utilizar.
   - *Classification*: Provee acceso a todas las técnicas de clasificación y regresión incluidas en Weka
   - *Cluster*: En este se integran varios métodos de agrupamiento.
   - *Associate*: Incluye las técnicas utilizadas para reglas de asociación.
   - *Select Attributes*: Contiene las opciones de acceso a las diversas técnicas usadas para la reducción de la cantidad o el numero de atributos
   - *Visualize*: Esta sección permite hacer estudios de comportamiento, usando las técnicas de visualización incluidas en Weka.

4. ***KnowledgeFlow***: Esta interfaz es flexible para generar proyectos de minería de datos mediante la generación de Flujos de Información.

Figura 2.2: Pantalla inicial de Weka.

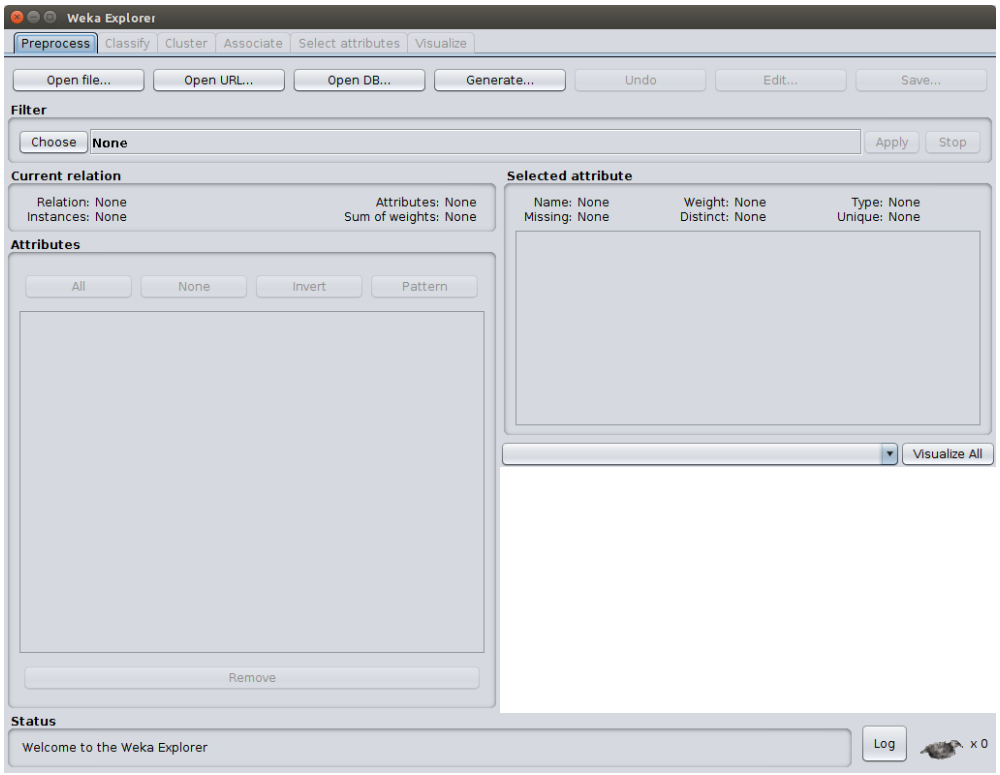


Figura 2.3: Interfaz del modulo *Explorer* de Weka.

2.1.2. SciKit-Learn

Figura 2.4: Logo del proyecto SciKit-Learn.

Es un paquete para el desarrollo de ML en Python. Provee herramientas simples y eficientes para la minería y el análisis de datos. Es accesible para cualquier persona y re-utilizable en varios contextos. Esta construida utilizando otras bien conocidas librerías de Python para el análisis numérico como lo son NumPy, SciPy y matplotlib. Es una librería de código abierto y que se

puede utilizar comercialmente bajo la licencia BSD.

Actualmente muchas empresas lo utilizan para implementar sus métodos de ML, ejemplos de estas: Spotify, Booking, Change.org, Evernote, entre otras.

Sus usos se dan en diversos campos como la visión artificial, procesamiento de lenguaje natural, traducción automática, robótica.

Incluye herramientas para clasificación, regresión, agrupamiento, reducción de dimensiones, selección de modelos y pre-procesado de datos.

2.1.3. Pandas

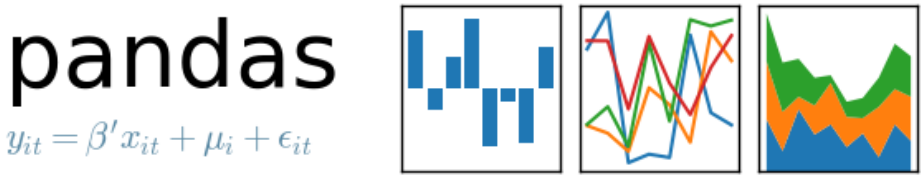


Figura 2.5: Logo del proyecto Pandas.

Pandas es una librería *open-source* la cual provee herramientas para el análisis de datos y estructuras de datos de fácil uso y alto rendimiento. Esta librería permite llevar a cabo el modelado y análisis de los datos, permitiendo tener todo el flujo de datos en una sola aplicación, evitando así la necesidad de un cambio a un lenguaje de dominio más especifico como R. Pandas funciona como un complemento a SciKit-Learn y a las demás librerías para trabajar con números, experimentos etc. que existen para Python.

2.2. Construcción de Clasificadores

2.2.1. Algoritmo de los vecinos más cercanos

Análisis de vecinos más próximos es un método para clasificar casos basándose en su parecido a otros casos. En el aprendizaje automático, se desarrolló como una forma de reconocer patrones de datos sin la necesidad de una coincidencia exacta con patrones o casos almacenados. Las instancias parecidos están próximas y las que no lo son están alejados entre sí. Por lo tanto, la distancia entre dos instancias es una medida de disimilitud. Las instancias próximas entre sí se denominan “vecinos”. Cuando se presenta una nueva instancia, se calcula su distancia con respecto a instancias de entrenamiento en el modelo. Las categorías de las instancias más parecidas (los vecinos más próximos) se evalúan y el nuevo caso se incluye en la categoría que contiene el mayor número de vecinos más próximos.

2.2.2. K-nn

Cuando a un algoritmo de Vecinos más cercanos se le especifica el número de vecinos más próximos que deben examinarse k; este método se denomina método de los k vecinos más cercanos. El método de los k vecinos más cercanos (en inglés, *k-nearest neighbors*, abreviado k-nn) es un método de clasificación supervisada que sirve para estimar la función de densidad $F(x/C_j)$ de las instancias x por cada clase c_j. Es uno de los algoritmos de clasificación más simples. Pero incluso con esta simplicidad, este puede generar resultados altamente competitivos. Es de fácil implementación en una gran cantidad de lenguajes de programación.

K-NN puede ser utilizado para problemas de clasificación y de regresión. Sin embargo, es más ampliamente utilizado en problemas de clasificación a nivel industrial. Es comúnmente utilizado por su facilidad de implementación y su bajo tiempo de calculo.

Este es un método de clasificación no paramétrico, que estima el valor de la función de densidad de probabilidad o directamente la probabilidad a posteriori de que un elemento x pertenezca a la clase c_j a partir de la información proporcionada por el conjunto de prototipos. En el proceso de aprendizaje no se hace ninguna suposición acerca de la distribución de las instancias.

En el reconocimiento de patrones, el algoritmo K-nn es usado como método de clasificación de objetos (elementos) basado en un entrenamiento mediante ejemplos cercanos en el espacio de los elementos. K-nn es un tipo de aprendizaje vago, donde la función se aproxima solo localmente y todo el cómputo es diferido a la clasificación.

2.2.2.1. Como funciona el algoritmo

Las instancias de entrenamiento son vectores en un espacio característico multidimensional, cada ejemplo está descrito en términos de p atributos considerando q clases para la clasificación. Los valores de los atributos del i-ésimo ejemplo (donde $1 \leq i \leq n$) se representan por el vector p-dimensional $x_i = (x_{1i}, x_{2i}, ..., x_{pi}) \in X$

El espacio es particionado en regiones por localizaciones y etiquetas de los ejemplos de entrenamiento. Un punto en el espacio es asignado a la clase C si esta es la clase más frecuente entre los k ejemplos de entrenamiento más cercanos. Generalmente se usa la distancia euclidiana.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{p}(x_{ri} - x_{rj})^2}$$

La fase de entrenamiento del algoritmo consiste en almacenar los vectores característicos y las etiquetas de las clases de los ejemplos de entrenamiento. En la fase de clasificación, la evaluación del ejemplo (del que no se conoce su clase) es representada por un vector en el espacio característico. Se calcula la distancia entre los vectores almacenados y el nuevo vector, y se seleccionan los k ejemplos más cercanos. El nuevo ejemplo es clasificado con la clase que más se repite en los vectores seleccionados (Figura 2.6).

Este método supone que los vecinos más cercanos nos dan la mejor clasificación y esto se hace utilizando todos los atributos; el problema de dicha suposición es que es posible que se tengan

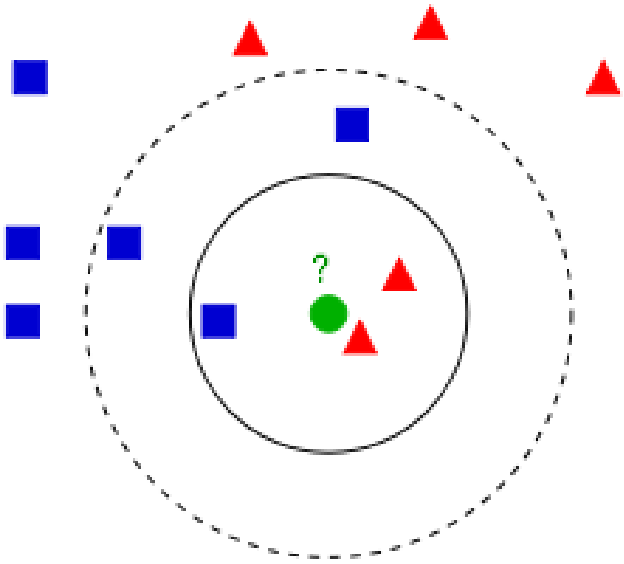


Figura 2.6: Supongamos que deseamos encontrar la categoría a la cual pertenece el circulo azul de la imagen, al seleccionar un $k = 3$, el algoritmo lo clasifica como perteneciente a la categoría del triangulo rojo. Por el contrario se seleccionamos un $k = 5$ la categoría asignada sera la del cuadrado azul. La elección del valor de k es crucial para el buen funcionamiento del algoritmo.

muchos atributos irrelevantes que dominen sobre la clasificación: dos atributos relevantes perderían peso entre otros veinte irrelevantes.

Para corregir el posible sesgo se puede asignar un peso a las distancias de cada atributo, dándole así mayor importancia a los atributos más relevantes. Otra posibilidad consiste en tratar de determinar o ajustar los pesos con ejemplos conocidos de entrenamiento. Finalmente, antes de asignar pesos es recomendable identificar y eliminar los atributos que se consideran irrelevantes. En síntesis, el método K-nn se resume en dos algoritmos: algoritmo de entrenamiento y algoritmo de clasificación.

2.2.2.1.1. Algoritmo de entrenamiento Para cada instancia $< x, f(x) >$, donde $x \in X$, agregar la instancia a la estructura representando las instancias de entrenamiento.

2.2.2.1.2. Algoritmo de clasificación Dado un ejemplar x_q que debe ser clasificado, sean $x_1, ...x_k$ los k vecinos más cercanos a x_q en las instancias de entrenamiento, se produce la siguiente salida:

$$\hat{f}(x) \leftarrow argmax_{v \in V} \sum_{i=1}^{k} (\delta(v, f(x_i))$$

donde $\delta(a, b) = 1$, si $a = b$ y 0 en cualquier otro caso. El valor $\hat{f}(x_q)$ devuelto por el algoritmo como un estimador de $f(x_q)$ es solo el valor más común de f entre los k vecinos más cercanos a x_q. Si elegimos $k = 1$; entonces el vecino más cercano a x_i determina su valor.

2.2.2.1.3. Elección del parámetro k La mejor elección de k depende fundamentalmente de los datos; generalmente, valores grandes de k reducen el efecto de ruido en la clasificación, pero crean límites entre clases parecidas. Un buen k puede ser seleccionado mediante una optimización de uso.

El caso especial en que la clase que se predice es la clase de la instancia de entrenamiento más cercano (cuando $k = 1$) es llamada *Nearest Neighbor Algorithm*, Algoritmo del vecino más cercano.

La exactitud de este algoritmo puede ser severamente degradada por la presencia de ruido o características irrelevantes, o si las escalas de características no son consistentes con lo que se considera importante para el problema. Muchas investigaciones y esfuerzos se han realizado para la selección de las características que mejoren las clasificaciones.

En la Figura 2.7 se puede observar la influencia de k en la construcción de las fronteras entre dos categorías. Se puede apreciar como a medida que se incrementa el valor de k, la frontera se suaviza. Si k tiende al infinito, se terminará teniendo toda la gráfica de color azul o rojo dependiendo de la categoría con más muestras en el conjunto.

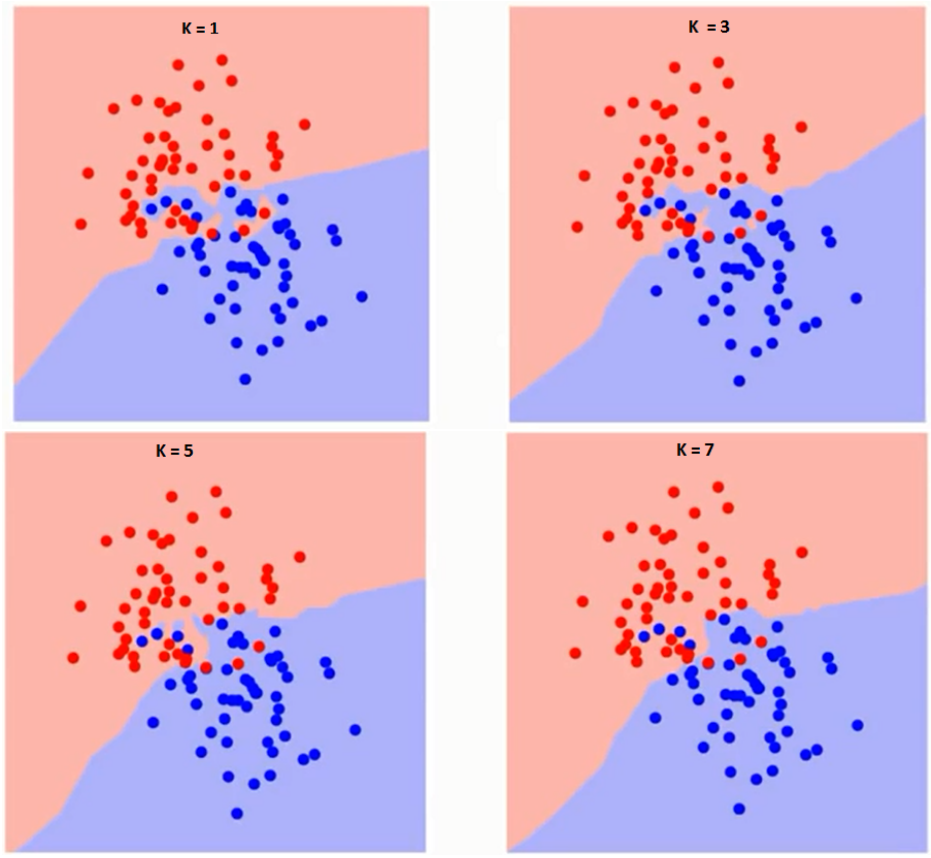


Figura 2.7: Efecto de k en la construcción del clasificador.

2.2.2.1.4. Variantes del K-nn

- K-nn con rechazo: se requieren garantías para clasificar
- K-nn con distancia media: se asigna el nuevo caso a la clase que tenga la distancia media menor.
- K-nn con distancia mínima: se selecciona el centroide de cada clase y el nuevo caso se asigna al que mas cerca este.

Ejemplos de Uso

- En los sistemas actuales de GPS
- Ubicación en redes sociales

2.2.3. Arboles de Decisión

Los arboles decisión son algoritmos versátiles de ML los cuales pueden ejecutar tareas de clasificación o regresión, incluso tareas que involucren múltiples salidas. Estos son algoritmos muy poderosos capaces de ajustarse a datasets complejos [Géron, 2017].

Un árbol de decisión es un modelo de predicción utilizado en diversos ámbitos que van desde la inteligencia artificial hasta la Economía. Dado un conjunto de datos se fabrican diagramas de construcciones lógicas, muy similares a los sistemas de predicción basados en reglas, que sirven para representar y categorizar una serie de condiciones que ocurren de forma sucesiva, para la resolución de un problema.

2.2.3.1. Elementos de un Árbol de decisión

Los árboles de decisión están formados por nodos, vectores de números, flechas y etiquetas. Cada nodo se puede definir como el momento en el que se ha de tomar una decisión de entre varias posibles, lo que va haciendo que a medida que aumenta el número de nodos aumente el número de posibles finales a los que puede llegar el individuo. Esto hace que un árbol con muchos nodos sea complicado de dibujar a mano y de analizar debido a la existencia de numerosos caminos que se pueden seguir. Los vectores de números serían la solución final a la que se llega en función de las diversas posibilidades que se tienen, dan las utilidades en esa solución. Las flechas son las uniones entre un nodo y otro y representan cada acción distinta. Las etiquetas se encuentran en cada nodo. Cada flecha da nombre a cada acción.

2.2.3.2. Reglas

En los árboles de decisión se tiene que cumplir una serie de reglas.

1. Al comienzo del juego se da un nodo inicial que no es apuntado por ninguna flecha, es el único del juego con esta característica.

2. El resto de nodos del juego son apuntados por una única flecha.

3. De esto se deduce que hay un único camino para llegar del nodo inicial a cada uno de los nodos del juego. No hay varias formas de llegar a la misma solución final, las decisiones son excluyentes.

En los árboles de decisiones las decisiones que se eligen son lineales, a medida que vas seleccionando entre varias opciones se van cerrando otras, lo que implica normalmente que no hay marcha atrás. En general se podría decir que las normas siguen una forma condicional: $Opción1 \rightarrow opción2 \rightarrow opción3 \rightarrow ResultadoFinalX$. Estas reglas suelen ir implícitas en el conjunto de datos a raíz del cual se construye el árbol de decisión.

Para comprender el funcionamiento de los arboles de decisión, podemos emplear el dataset iris, el cual contiene la información de varios tipos de flores. La información en este dataset incluye la longitud y anchura de los pétalos y sépalos de 3 tipos de flores diferentes.

Cuadro 2.1: Muestras de instancias del Dataset Iris

Largo de sépalo	**Ancho de sépalo**	**Largo de pétalo**	**Ancho de pétalo**	**Especies**
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
4.7	3.2	1.3	0.2	I. setosa
5.3	3.7	1.5	0.2	I. setosa
5.0	3.3	1.4	0.2	I. setosa
7.0	3.2	4.7	1.4	I. versicolor
6.4	3.2	4.5	1.5	I. versicolor
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
5.1	2.5	3.0	1.1	I. versicolor
5.7	2.8	4.1	1.3	I. versicolor
6.3	3.3	6.0	2.5	I. virginica
5.8	2.7	5.1	1.9	I. virginica
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
7.1	3.0	5.9	2.1	I. virginica
6.3	2.9	5.6	1.8	I. virginica

Utilizando esta información se construye un árbol de decisión, este árbol esta compuesto de nodos a diferentes niveles de profundidad Figura 2.9. En el supuesto de que encontramos una flor iris y deseamos saber que tipo de flor es, iniciaríamos nuestro análisis en el nivel 0 del árbol (parte superior), este nodo pregunta si la longitud del pétalo de la flor es menor a 2,45 cm. Si es este el caso, nos movemos del nodo raíz del nivel 0 hacia el hijo izquierdo del nodo (nivel 1). En este caso este nodo es un nodo hoja (no tiene hijos), por lo cual la categoría para nuestra flor sera la indicada por este nodo hoja. En este caso *categoría = setosa*.

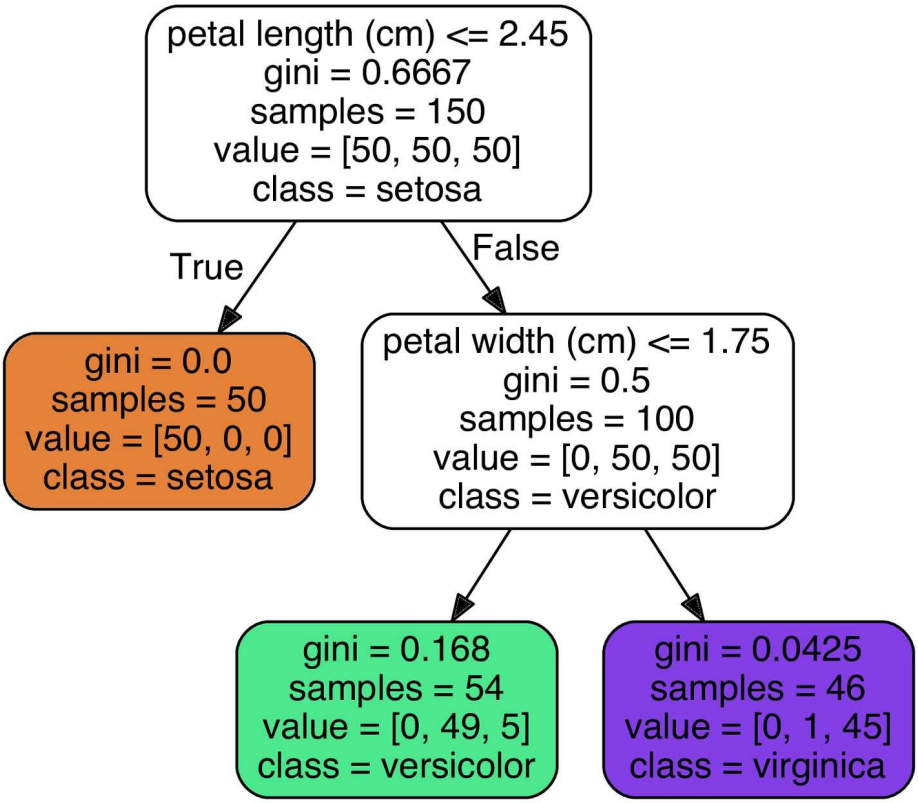


Figura 2.8: Árbol de Decisión para el dataset Iris[Géron, 2017].

Ahora podemos suponer que encontramos otra flor, pero en este caso la longitud del pétalo es mayor a 2,45 cm, por lo tanto debemos movernos hacia el hijo izquierdo del nodo raíz. Este, no es un nodo hoja, por lo cual es un nodo que realiza una pregunta: ¿Es el ancho del pétalo menor que 1,75 cm? Si esta condición se cumple, entonces nos moveremos hacia el hijo izquierdo de este nodo, por lo cual categoría seria Iris Versicolor. Si el ancho del pétalo es mayor a 1,75 cm, entonces nos movemos al hijo derecho del nodo, lo cual indicaría que es más probable que sea una flor Iris-Verginica.

Los nodos tienen atributos, el atributo muestras/instancias cuenta la cantidad de instancias que cumple con la condición que evaluá el nodo, el atributo valor nos dice cuantas instancias de

entrenamiento de cada categoría, cumplen con la condición del nodo, en este atributo, se tiene un vector en el que cada dimensión corresponde con una categoría. La suma de los valores de cada dimensión se corresponde con el atributo muestras. El tercer atributo de un nodo es el gini, el cual mide como de puro es un nodo, un $gini = 0$ significa que todas las instancias que cumplieron la condición evaluada por el nodo, pertenecen a la misma categoría. Para el clasificador de flores, el nodo de nivel 1-derecha, obtiene una cantidad de 46 instancias, en el atributo valor se obtiene el vector $valor = [0, 50, 50]$ lo cual indica que, de las 100 muestras con longitud de pétalo mayor a $2,45$ cm, 50 pertenecen a la categoría Iris-Versicolor y 50 instancias a la categoría Iris-Virginica. El valor gini del nodo seria $1 - (\frac{0}{100})^2 - (\frac{50}{100})^2 - (\frac{50}{100})^2 = 0{,}5$

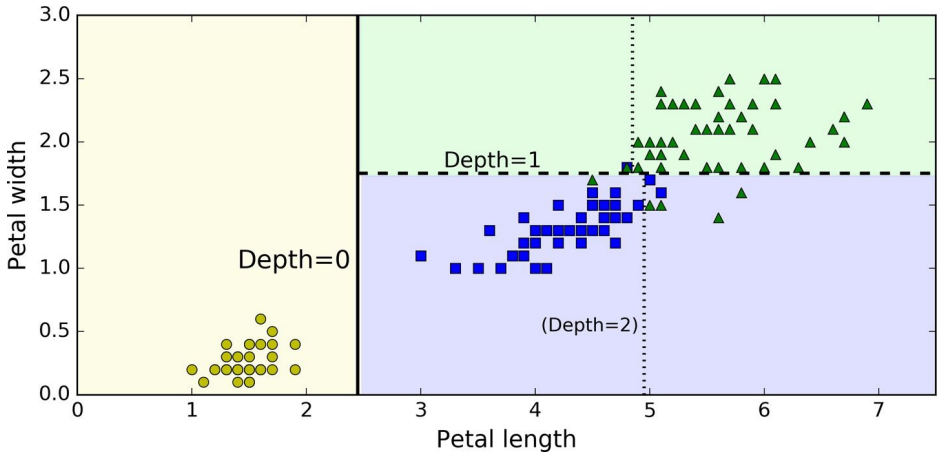


Figura 2.9: Fronteras del árbol de decisión[Géron, 2017].

En la imagen se aprecian lo limites del árbol de decisión. La linea vertical representa el limite del nodo base(nivel 0). Debido a la pureza de este nodo, ya no se puede dividir posteriormente (solo contiene instancias de la categoría Iris-Setosa). Sin embargo el lado derecho es impuro, por lo cual el nodo en este nivel (1) divide las muestras según la anchura del pétalo. Este genera 2 nodos hojas, creando el nivel 2 de profundidad, esto se representa con los guiones . Si se estableciera la profundidad el árbol en 3, se generaría un nuevo nivel, el cual se representa con la línea punteada.

Los arboles de decisión pueden también predecir probabilidades de pertenencia a cada categoría del dataset, para una muestra que se este evaluando. Si suponemos encontrar una flor con pétalos de 5 cm de largo y 1,5 cm de ancho. Su nodo correspondiente seria el del nivel 2 izquierda. Por lo tanto la salida estará compuesta de las probabilidades expresadas en la Tabla 2.2. Pero si se le preguntase por la categoría su salida seria Iris-Versicolor.

2.2.4. Random Forest

Random Forests es una técnica de agregación desarrollada por Leo Breiman, que mejora la precisión en la clasificación mediante la incorporación de aleatoriedad en la construcción de cada

Cuadro 2.2: Probabilidades Obtenidas para la muestra.

Probabilidad	Categoría	Distribución
0 %	Iris-Setosa	(0/54)
90,7 %	Iris-Versicolor	(49/54)
9,3 %	Iris-Virginica	(5/54)

clasificador individual. Esta aleatorización puede introducirse en la partición del espacio (construcción del árbol), así como en la muestra de entrenamiento.

Como se ha comentado anteriormente un Random Forest es conjunto de arboles de decisión. Estos arboles son generalmente entrenados mediante algún método de bagging. El bagging (Figura 2.10) consiste en entrenar el mismo algoritmo de clasificación pero utilizando solo una parte del conjunto de entrenamiento. Una vez que cada modelo es entrenado, se unen y la categoría para una instancia en evaluación se obtiene mediante la totalización de los resultados generados por cada modelo. Generalmente esta totalización es en un modo estadístico, por lo cual gana la categoría más votada o la de mayor frecuencia entre las predicciones de los modelos.

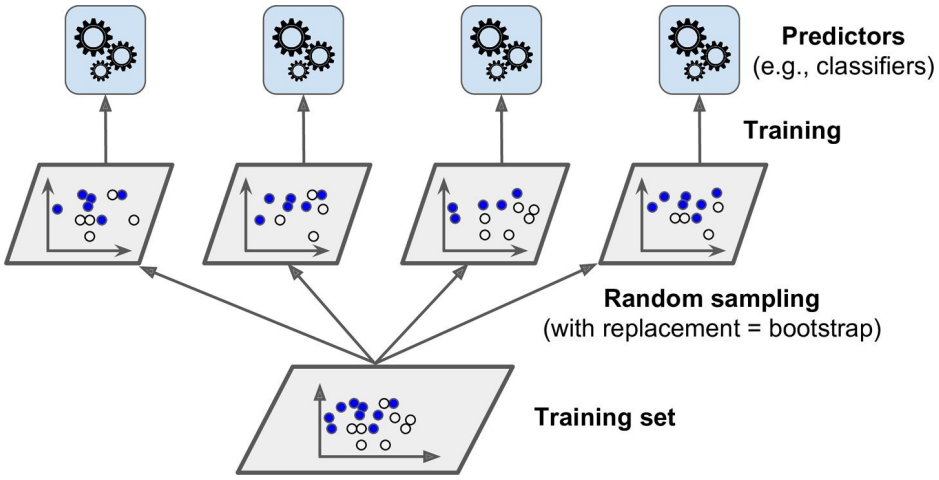


Figura 2.10: Muestreo de Conjuntos de Entrenamiento para bagging y entrenamiento del clasificador[Géron, 2017].

El algoritmo Random Forest esta más optimizado que los arboles de decisión. El uso de *Random Forest* añade cierto nivel de aleatoridad en el momento en que se construye el árbol; en lugar de buscar la mejor característica cuando se hace la división en algún nodo, el algoritmo busca la mejor característica pero en un subconjunto de los datos de entrenamiento. Esto al final genera un árbol de gran diversidad .

2.2.5. Agrupamiento Jerárquico

El agrupamiento jerárquico es un método de análisis de grupos puntuales, el cual busca construir una jerarquía de grupos. Estrategias para agrupamiento jerárquico generalmente caen en dos tipos:

- Aglomerativas: Este es un acercamiento ascendente: cada observación comienza en su propio grupo, y los pares de grupos son mezclados mientras uno sube en la jerarquía.

- Divisivas: Este es un acercamiento descendente: todas las observaciones comienzan en un grupo, y se realizan divisiones mientras uno baja en la jerarquía.

Los resultados del agrupamiento jerárquico son usualmente presentados en un dendrograma. Las Figuras 2.11 y 2.12, muestran el antes y después respectivamente, luego de la aplicación del Algoritmo.

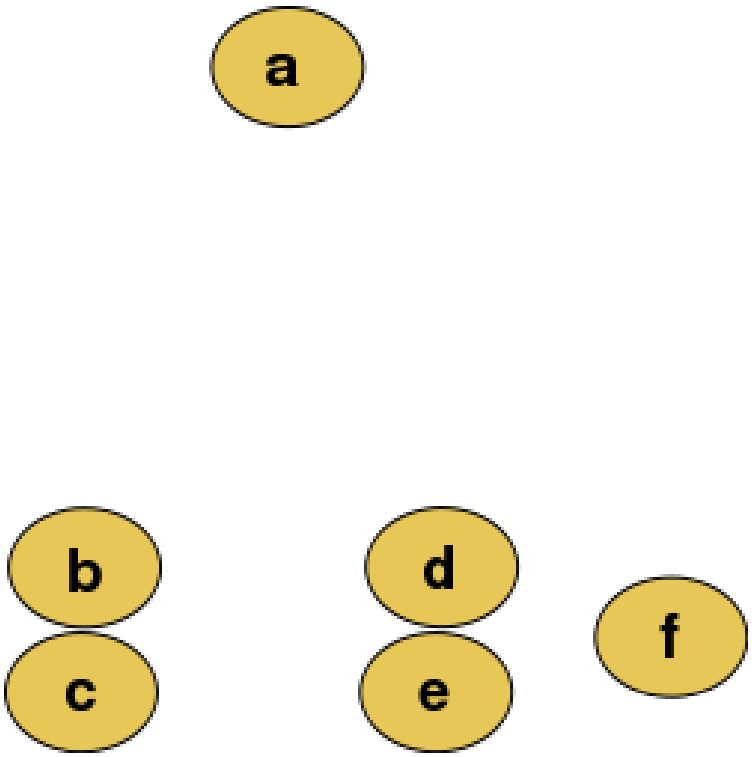


Figura 2.11: Inicio de las muestras antes de la ejecución del algoritmo.

En el caso general, la complejidad del agrupamiento aglomerativo es $\mathcal{O}(n^3)$, lo cual los hace demasiado lentos para grandes conjuntos de datos. El agrupamiento divisivo con búsqueda exhaustiva es $\mathcal{O}(2^n)$ lo cual es aún peor. Sin embargo existen métodos óptimos y eficientes para los algoritmos aglomerativos.

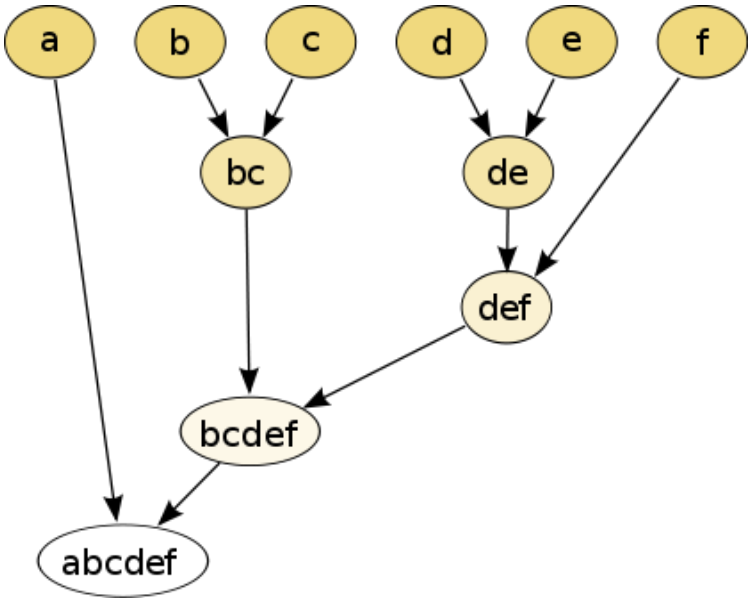


Figura 2.12: Grupos creados por el algoritmo de agrupamiento jerárquico.

2.2.5.1. Disimilitud de grupo

En orden de decidir qué grupos deberían ser combinados (para aglomerativo), o cuando un grupo debería ser dividido (para divisivo), una medida de disimilitud entre conjuntos de observaciones es requerida. En la mayoría de los métodos de agrupamiento jerárquico, esto es logrado mediante uso de una métrica apropiada (una medida de distancia entre pares de observaciones), y un criterio de enlace el cual especifica la disimilitud de conjuntos como una función de las distancias dos a dos entre observaciones en los conjuntos.

2.2.5.2. Métrica

La elección de una métrica apropiada influenciará la forma de los grupos, ya que algunos pueden estar cerca unos de otros de acuerdo a una distancia y más lejos de acuerdo a otra. Por ejemplo, en un espacio 2-dimensional, la distancia entre el punto $(1, 0)$ y el origen $(0, 0)$ es siempre 1 de acuerdo a las normas usuales, pero la distancia entre el punto $(1, 1)$ y el origen $(0, 0)$ puede ser 2, $\sqrt{2}$, o 1 bajo la distancia Manhattan, la distancia euclidiana o la distancia máxima respectivamente.

Algunas métricas comúnmente usadas para agrupamiento jerárquico son: Distancia Euclidiana, Distancia Euclidiana al cuadrado, Manhatan, distancia máxima, Mahalanobis o Similitud coseno.

2.2.5.3. Criterio de enlace

El criterio de enlace determina la distancia entre conjuntos de observaciones como una función de las distancias entre observaciones dos a dos. Algunos criterios de enlace entre dos conjuntos de

observaciones A y B frecuentemente usados son: enlace completo, enlace simple, media o mínima energía.

El agrupamiento jerárquico tiene la ventaja distintiva de que cualquier medida de distancia puede ser usada. De hecho, las observaciones de por si no son requeridas: todo lo que se usa es una matriz de distancia.

2.2.6. K-medias

El proceso de agrupamiento k-medias es particional, es decir, forma subgrupos a partir de un grupo más general, inicialmente se determina el número de grupos K que se desean formar y se eligen los centroides. Para determinar los centroides iniciales, hay dos alternativas: la primera es tomar de forma aleatoria K objetos como centroides iniciales y la segunda es tomar los primeros K objetos del conjunto de objetos que se estén analizando.

Tiene como objetivo la partición de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Es un método utilizado en minería de datos.

La agrupación del conjunto de datos puede ilustrarse en una partición del espacio de datos en celdas de Voronoi.

El problema es computacionalmente difícil. Sin embargo, hay eficientes heurísticas que se emplean comúnmente y convergen rápidamente a un óptimo local. Estos suelen ser similares a los algoritmos *expectation-maximization* de mezclas de distribuciones gausianas por medio de un enfoque de refinamiento iterativo empleado por ambos algoritmos. Además, los dos algoritmos usan los centros que los grupos utilizan para modelar los datos, sin embargo k-means tiende a encontrar grupos de extensión espacial comparable, mientras que el mecanismo *expectation-maximization* permite que los grupos tengan formas diferentes.

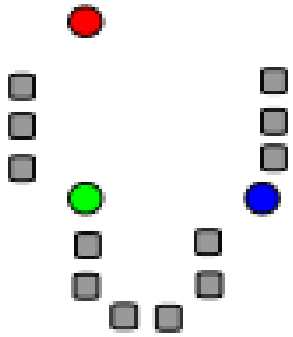
2.2.6.1. Descripción

Dado un conjunto de observaciones $x_1, x_2, ..., x_n$, donde cada observación es un vector real de d dimensiones, k-means construye una partición de las observaciones en k conjuntos (kn) a fin de minimizar la suma de los cuadrados dentro de cada grupo $S = S_1, S_2, ..., S_k$. La Figura 2.13 muestra los pasos iniciales del algoritmo.

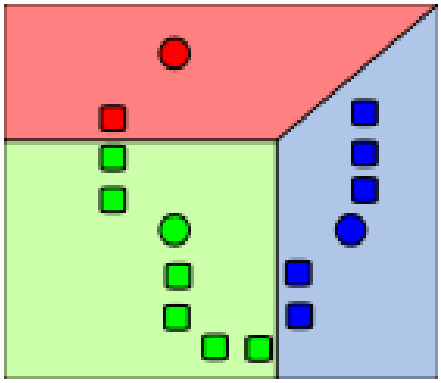
$$\underset{S}{argmin} \sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

donde μ_j es la media de los puntos en S_j.

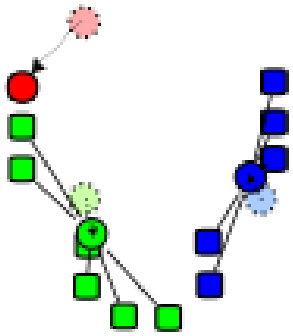
Como se trata de un algoritmo heurístico, no hay ninguna garantía de que convergen al óptimo global, y el resultado puede depender de los grupos iniciales. Como el algoritmo suele ser muy rápido, es común para ejecutar varias veces con diferentes condiciones de partida. Sin embargo, en el peor de los casos, k-means puede ser muy lento para converger:



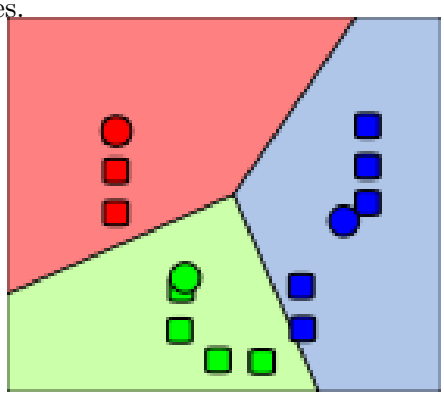
1) k centroides iniciales (en este caso $k = 3$) son generados aleatoriamente dentro de un conjunto de datos (mostrados en color).



2) k grupos son generados asociándole el punto con la media más cercana. La partición aquí representa el diagrama de Voronoi generado por los centroides.



3) El centroide de cada uno de los k grupos se recalcula.



4) Pasos 2 y 3 se repiten hasta que se logre la convergencia.

Figura 2.13: Funcionamiento del Algoritmo k-means

2.2.7. Maquinas de Vectores de Soporte (Support Vector Machines) (SVM)

Un SVM es un poderoso y versátil modelo de ML, es capaz de ejecutar clasificaciones lineales o no lineales, regresión e incluso detección de *outliers*. Es uno de los modelos más populares para el ML, por lo cual cualquier persona interesada en el ML debe tenerlo en su conjunto de herramientas. Estos son adecuados para la clasificación de datasets complejos de pequeño o mediano tamaño. Tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 por Vapnik y sus colaboradores [Boser et al., 1992, Cortes & Vapnik, 1995]. Aunque originariamente las SVMs fueron pensadas para resolver problemas de clasificación binaria, actualmente se utilizan para resolver otros tipos de problemas (regresión, agrupamiento, multiclasificación). También son diversos los campos en los que han sido utilizadas con éxito, tales como visión artificial, reconocimiento de caracteres, categorización de texto e hipertexto, clasificación

de proteínas, procesamiento de lenguaje natural, análisis de series temporales. De hecho, desde su introducción, han ido ganando un merecido reconocimiento gracias a sus sólidos fundamentos teóricos.
Los SVM mapean los puntos de entrada a un espacio de características de una dimensión mayor, para luego encontrar el hiperplano que los separe y maximice el margen entre las clases. Pertenecen a la familia de clasificadores lineales puesto que inducen separadores lineales o hiperplanos en espacios de características de muy alta dimensionalidad (introducidos por funciones núcleo o *kernel*) con un sesgo inductivo muy particular

La formulación matemática de las SVM varía dependiendo de la naturaleza de los datos; es decir, existe una formulación para los casos lineales y, por otro lado, una formulación para casos no lineales. Es importante tener claro que, de manera general para clasificación, las SVM buscan encontrar un hiperplano óptimo que separe las clases. Las SVM han sido desarrolladas como una técnica robusta para clasificación y regresión aplicado a grandes conjuntos de datos complejos con ruido; es decir, con variables inherentes al modelo que para otras técnicas aumentan la posibilidad de error en los resultados pues resultan difíciles de cuantificar y observar.

2.2.7.1. Idea básica

Dado un conjunto de puntos, subconjunto de un conjunto mayor (espacio), en el que cada uno de ellos pertenece a una de dos posibles categorías, un algoritmo basado en SVM construye un modelo capaz de predecir si un punto nuevo (cuya categoría desconocemos) pertenece a una categoría o a la otra. Como en la mayoría de los métodos de clasificación supervisada, los datos de entrada (los puntos) son vistos como un vector p-dimensional (una lista ordenada de p números). Un SVM busca un hiperplano que separe de forma óptima a los puntos de una clase de la de otra, que eventualmente han podido ser previamente proyectados a un espacio de dimensionalidad superior(Figura 2.14).

En ese concepto de "separación óptima" es donde reside la característica fundamental de las SVM: este tipo de algoritmos buscan el hiperplano que tenga la máxima distancia (margen) con los puntos que estén más cerca de él mismo. Por eso también a veces se les conoce a las SVM como clasificadores de margen máximo. De esta forma, los puntos del vector que son etiquetados con una categoría estarán a un lado del hiperplano y los casos que se encuentren en la otra categoría estarán al otro lado. Los algoritmos SVM pertenecen a la familia de los clasificadores lineales. También pueden ser considerados un caso especial de la regularización de Tikhonov.

En la literatura de los SVMs, se llama atributo a la variable predictora y característica a un atributo transformado que es usado para definir el hiperplano. La elección de la representación más adecuada del universo estudiado, se realiza mediante un proceso denominado selección de características. Se denominan vectores de soporte a los puntos que conforman las dos líneas paralelas al hiperplano, siendo la distancia entre ellas (margen) la mayor posible. Los modelos basados en SVMs están estrechamente relacionados con las redes neuronales. Usando una función kernel, re-

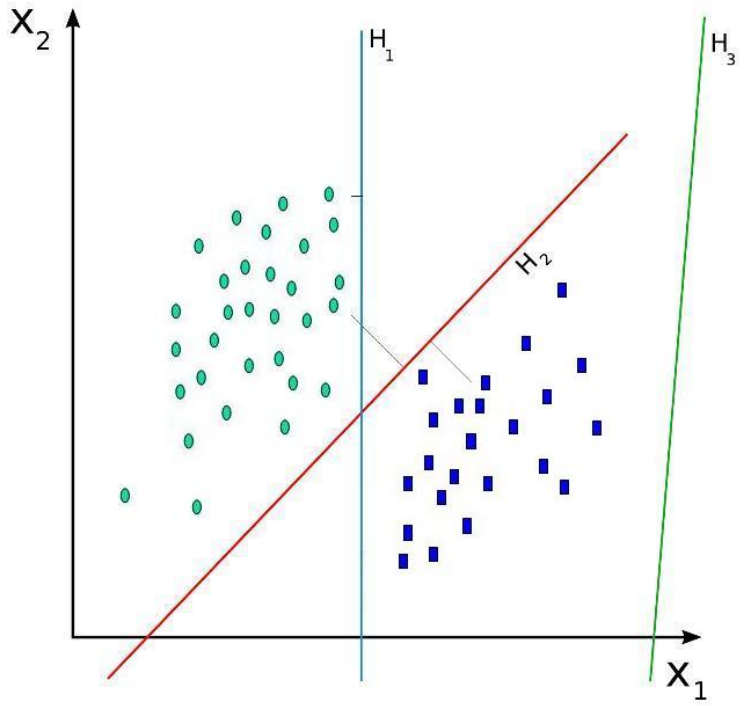


Figura 2.14: Hipótesis de separación de un conjunto de datos.

sultan un método de entrenamiento alternativo para clasificadores polinomiales, funciones de base radial y perceptrón multicapa.

2.2.7.2. SVM de Clasificacion lineal

La idea principal detrás de un SVM se explica mejor utilizando imágenes 2.15. En estas se muestra parte del dataset Iris. Estas dos clases pueden ser fácilmente separadas con una linea recta (son linealmente separables). La gráfica de la izquierda muestra los limites de decisión para tres posibles clasificadores lineales diferentes. El modelo cuya linea se muestra punteada es malo, ya que no puede separar las dos clases apropiadamente. Los demás modelos trabajan perfectamente en este conjunto de entrenamiento, pero sus limites de decisión están demasiado cerca a las instancias que estos modelos probablemente no funcionen tan adecuadamente cuando se evalúan nuevas instancias. Por lo contrario la linea solida en la gráfica de la derecha representa el limite de decisión de un clasificador SVM, esta linea no solo separa las dos clases sino que también se mantiene lo más alejado posible de las instancias de clases mas cercanas. Se puede pensar en el SVM como ajustándose a la calle mas ancha posible (representada por las lineas punteadas) entre las clases. Esto se llama clasificación de margen amplio.

Si se añaden mas instancias de entrenamiento “fuera de la calle” esto no afectará el limite de decisión: este está completamente determinado o soportado por las instancias localizadas en los

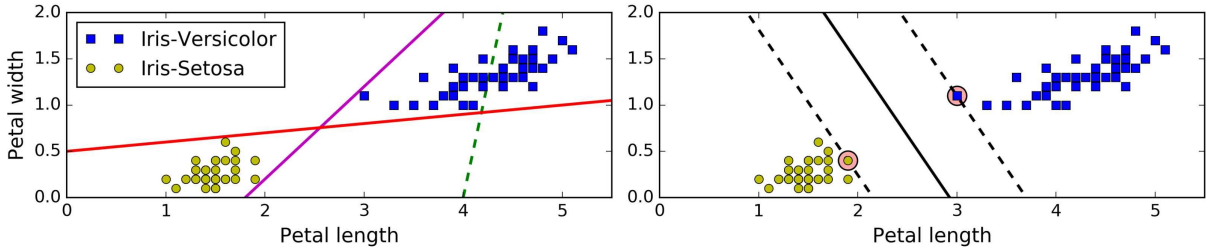


Figura 2.15: Clasificación de margen amplio[Géron, 2017].

bordes de la calle. Estas instancias reciben el nombre de vectores de soporte.

2.2.7.2.1. Clasificación de margen flexible Si estrictamente imponemos que todas las instancias estén fuera de la calle y en el lado derecho, eso se denomina clasificación de margen fijo. Existen dos tipos de consideraciones para este tipo de clasificación. Primero, esta solo trabaja si los datos son linealmente separables y segundo esta es bastante sensitiva a los *outliers*. En la Figura 2.16 se muestra el dataset Iris con un *outlier* adicional: a la izquierda, es imposible encontrar un margen fijo y en la derecha el limite termina bastante diferente al que se ve en la figura anterior.

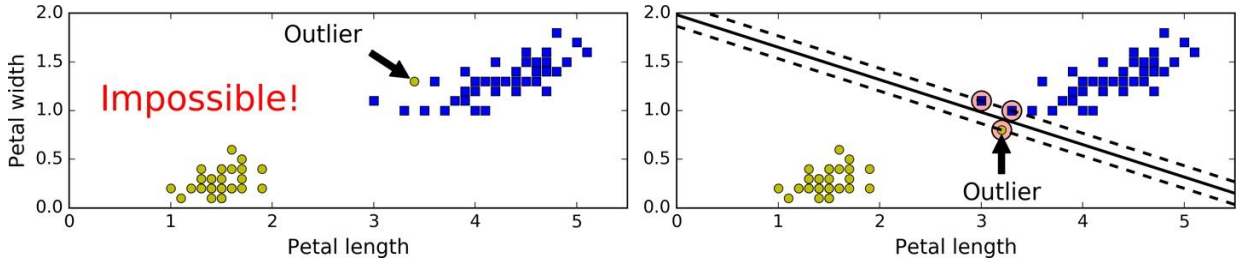


Figura 2.16: Un margen fijo es sensitivo a los *outliers*[Géron, 2017].

Para evitar estas dificultades es preferible utilizar un modelo más flexible. El objetivo es encontrar un buen balance entre mantener la calle lo más largo posible y limitar las violaciones del margen (instancias que terminan en la mitad de la calle o incluso en el lado equivocado). Esto se llama clasificación de margen flexible.

2.2.7.3. Clasificación No-Lineal con SVM

A pesar de que los clasificadores SVM son eficientes y funcionan relativamente bien en la mayoría de los casos, muchos datasets no están cerca de ser linealmente separables. Un enfoque para manejar un dataset no lineal es añadir más características, como características polinominales y en algunos casos esto resultará en un dataset linealmente separable. Como se observa en el lado izquierdo Figura 2.17 esto representa un dataset simple con solo una característica x_1 el cual no

es linealmente separable como se puede apreciar. Pero si se añade una segunda característica $x_2 = (x_1)^2$, el dataset resultante 2D es perfectamente separable. El algoritmo SVM trata de encontrar un hiperplano 1-dimensional (en el ejemplo es una línea) que une a las variables predictoras y constituye el límite que define si un elemento de entrada pertenece a una categoría o a la otra. Existe un número infinito de posibles hiperplanos (líneas) que realicen la clasificación pero, ¿cuál es la mejor y cómo la definimos? La mejor solución es aquella que permita un margen máximo entre los elementos de las dos categorías.

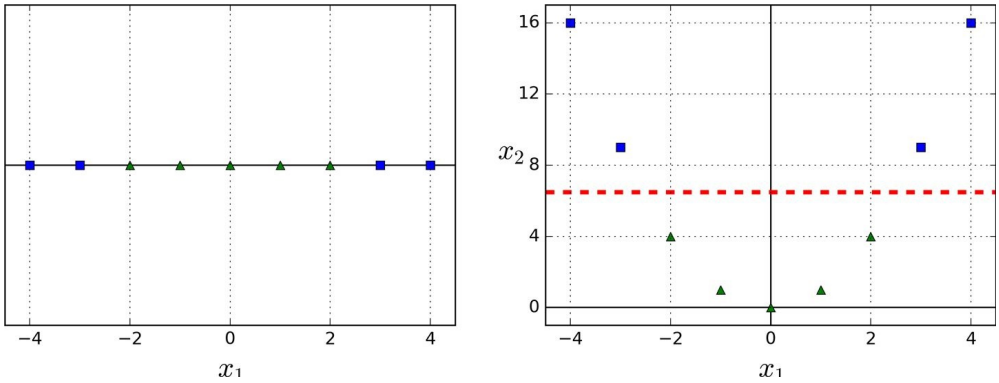


Figura 2.17: Agregación de características a un dataset para hacerlo linealmente separable[Géron, 2017].

2.2.7.3.1. *Kernels* Cuando los datos no se pueden separar linealmente. Se emplea el truco del *kernel*, el cual permite operar en un espacio de grandes dimensiones sin la necesidad de calcular las coordenadas de los datos en dicho espacio, sino simplemente calculado los productos escalares entre las imágenes de todo tipo de datos en el espacio de características.

2.2.7.3.1.1. *Kernel* Polinomial Este *Kernel* añade características polinomiales al dataset. Si se usan un polinomio de bajo grado este no podrá manejar datasets complejos y un polinomio de alto grado añade un gran numero de características lo cual hará que el modelo sea lento. Afortunadamente para los SVM se puede emplear una técnica llamada Kernel, la cual hace posible obtener el mismo resultado de si se hubiesen añadido muchas características polinomiales, incluso con polinomios de alto grado, pero sin haberlas agregado en realidad. De esta forma no hay una explosión combinacional del numero de características, debido a que realmente no se añaden ninguna característica (Figura 2.18).

2.2.7.3.1.2. *Kernel* RBF De igual manera que el *kernel* polinominal añade características polinomiales, un *kernel* RBF (*radial basic function*) añade características que se computan usando una función de similitud, la cual calcula cuanto se parece cada instancia a un punto de referencia. Usando este *kernel* se pueden obtener los resultados de añadir una gran numero de características de similitud, pero sin haberlas agregado (Figura 2.19).

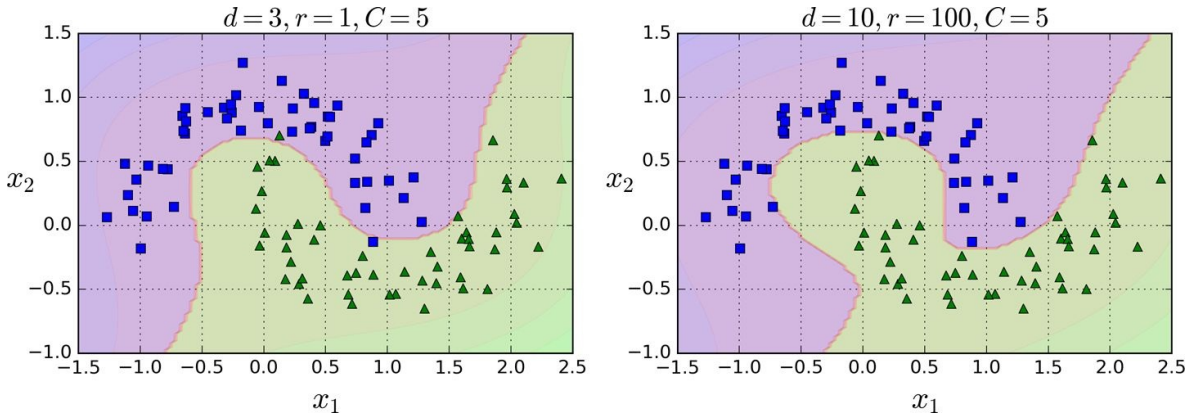


Figura 2.18: Clasificadores SVM utilizando el *kernel* Polinomial[Géron, 2017].

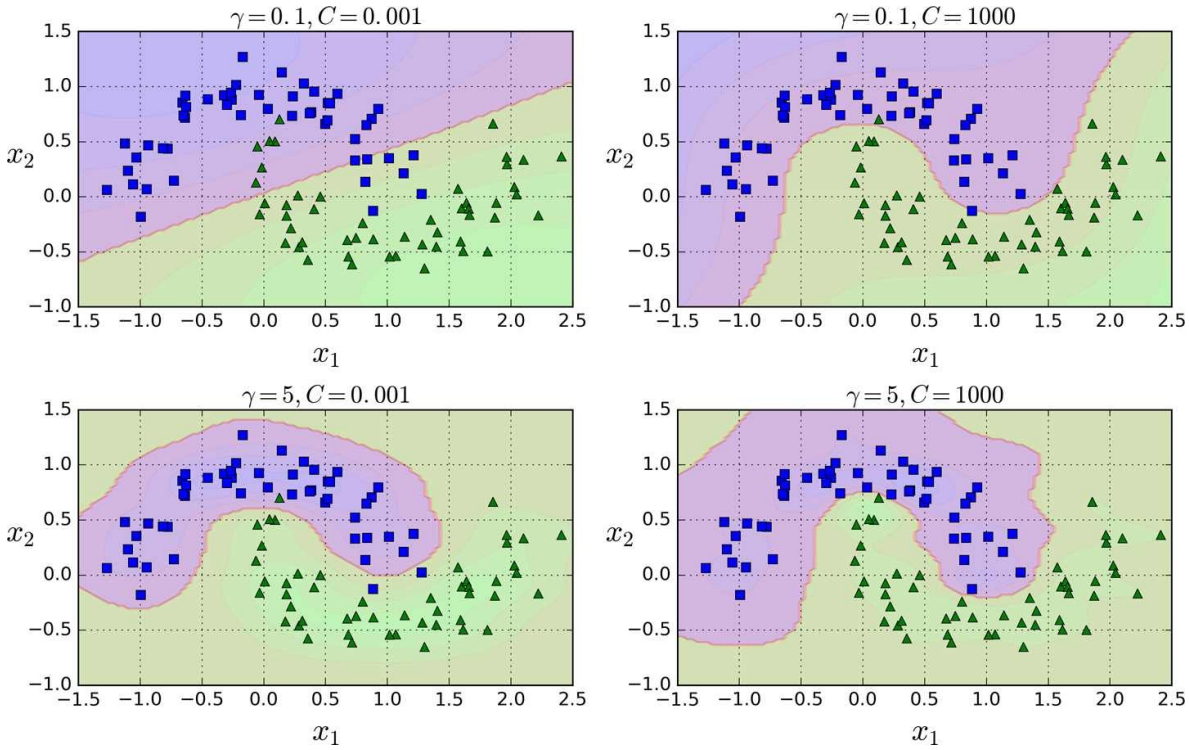


Figura 2.19: Clasificadores SVM utilizando el *kernel* RBF[Géron, 2017].

Parte II

Deep Learning

Capítulo 3

Deep Learning y Redes Neuronales Convolucionales

3.1. *Deep Learning*

3.1.1. Historia

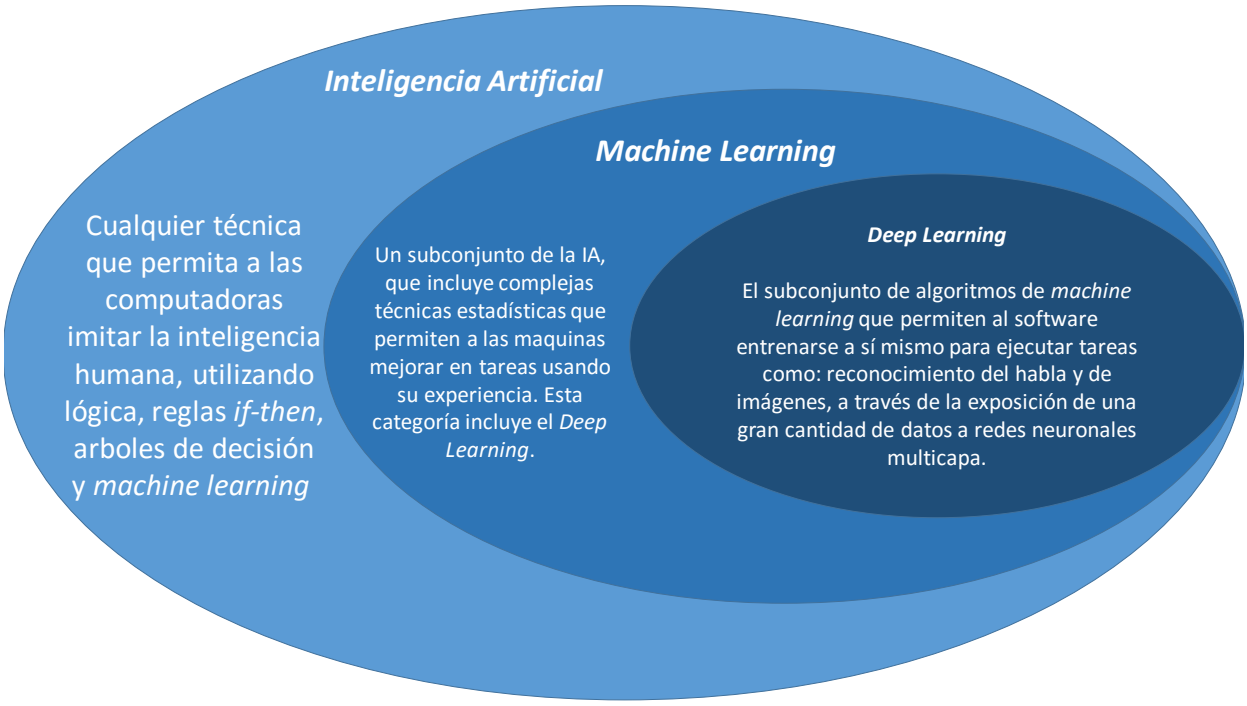


Figura 3.1: Definicion del *Deep Learning*.

El aprendizaje profundo o *Deep Learning* es una subárea del Aprendizaje Automático que usa la "redes Neuronales profundas", es decir, equipado con muchas capas y nuevos algoritmos para el procesamiento previo de datos para la regularización del modelo: incrustaciones de palabras,

abandonos, aumento de datos, etc. El aprendizaje profundo se inspira en la neurociencia ya que las redes neuronales son un modelo de actividad neuronal cerebral. A diferencia del cerebro biológico, donde cualquier neurona se puede conectar a cualquier otra neurona bajo ciertas restricciones físicas, las Redes Neuronales Artificiales (ANN) tienen un número finito de capas y conexiones, y finalmente tienen una dirección predeterminada de propagación de la información. Hasta ahora, las redes neuronales artificiales habian sido ignoradas por la comunidad investigadora por el principal problema de la industria, su costo computacional. Sin embargo, entre 2006 y 2012, el equipo de investigación dirigido por Geoffrey Hinton, de la Universidad de Toronto fue finalmente capaz de paralelizar los algoritmos para la ANN en arquitecturas paralelas. El resultado principal fue un aumento significativo en el número de capas, neuronas y parámetros del modelo en general (incluso más de 10 millones de parámetros) que permite a las máquinas calcular una cantidad masiva de datos entrenándose en él.

Entonces, el primer requerimiento para entrenar un modelo de Deep Learning es una base de datos muy cuantiosa para que él mismo se entrene. Por esto es importante la figura del *Big Data*.

Esto es porque existe un gran número de parámetros que necesitan ser entendidos por un algoritmo de aprendizaje, el cual inicialmente puede producir muchos falsos-positivos. La razón tras la popularidad del *Deep Learning* son el *Big Data* y las Unidades de procesamiento grafico (en inglés, *Graphic Processing Unit* (GPUs). Usando una gran porción de datos, los algoritmos y la red aprenden a cómo conseguir metas y mejorar sobre el proceso.

Ahora bien, tenemos que tener en cuenta que el *Deep Learning* es altamente susceptible al sesgo. Por lo tanto, del modo en el que el modelo de DL pueda aprender mejor que los humanos en un mismo sentido, puede estar equivocado. En un modelo supervisado si las etiquetas (*labels*) son establecidas de una manera incorrecta, el modelo va a aprender de los errores. Cuando el sistema de reconocimiento facial de Google fue inicialmente dado a conocer, por ejemplo, tageó muchas caras negras cómo gorilas. “Esto es un ejemplo de lo que sucede si no existen caras Afro Americanas en tu colección de entrenamiento”.

3.1.2. Redes Neuronales Artificiales

3.1.3. Perceptrón

El perceptrón [Géron, 2017] es uno de las arquitecturas de redes neuronales artificiales mas simples. Diseñada en 1957 por Frank Rosenblatt, esta basado en una ligeramente diferente neurona artificial llamada *linear threshold unit* (LTU)(ver Figura 3.2), sus entradas y salidas son números y cada conexión de entrada está asociada con un peso. La LTU calcula una suma ponderada de sus entradas ($z = w_1x_1 + w_2x_2 + \cdots w_nx_n = w^T \cdot x$), luego aplica una función de paso a este resultado y produce el resultado: $h_w(x) = step(z) = step(w^T \cdot x)$. Un ejemplo de función de paso (Heaviside) se aprecia en la ecuación 3.1 y la función SGN en la ecuación 3.2.

$$heaviside(z) = \begin{cases} 0 & si\ z < 0 \\ 1 & si\ z \geq 0 \end{cases} \tag{3.1}$$

$$sgn(z) = \begin{cases} -1 & si\ z < 0 \\ 0 & si\ z = 0 \\ 1 & si\ z > 0 \end{cases} \tag{3.2}$$

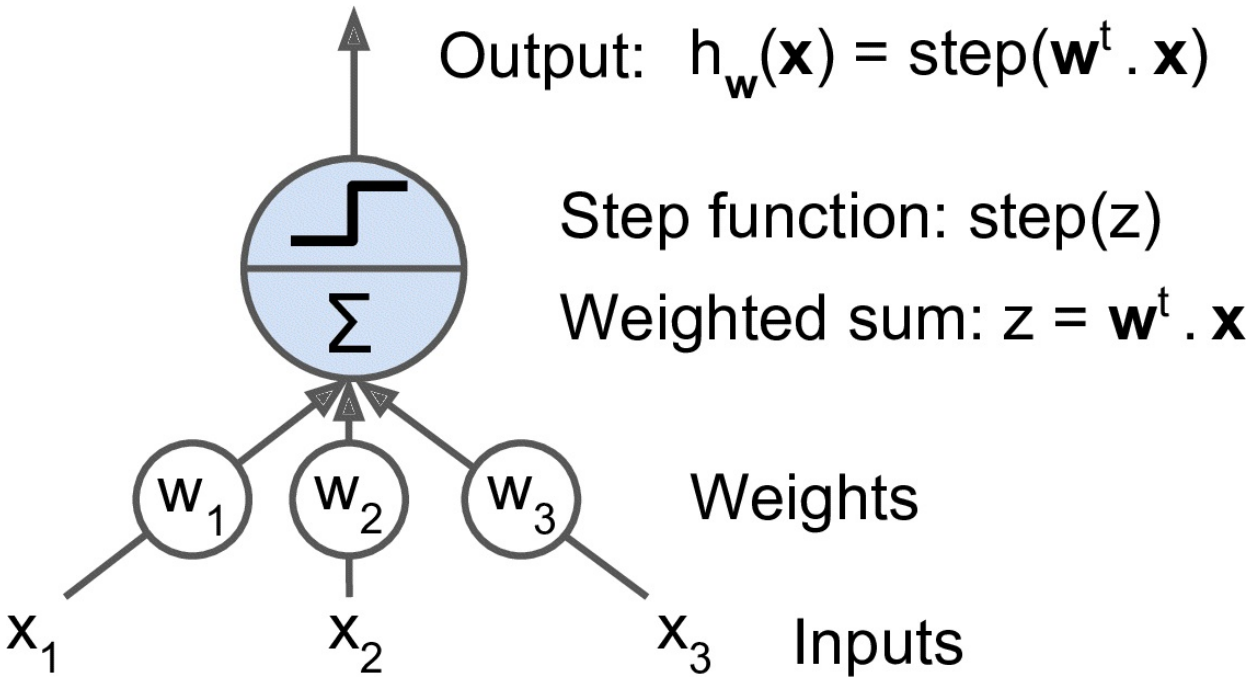


Figura 3.2: Diagrama de un LTU[Géron, 2017].

Una LTU simple se puede utilizar para clasificación lineal binaria. Este calcula la combinación lineal de las salidas y si su resultado excede un umbral la salida sera positiva de lo contrario sera negativa. Entrenar un perceptrón significa encontrar los valores adecuados para w_0, $w_1 0$ y w_2, para el perceptrón de la Figura 3.2.

El perceptrón esta compuesto de una simple capa de LTU's, con cada neurona conectada a todas las entradas. Estas conexiones son a menudo representadas utilizando un neurona de paso especial llamada neurona de entrada, estas solo generan como salida cualquier dato que le ha sido ingresado. Ademas, un característica extra llamada *bias* es generalmente agregada ($x_0 = 1$). Este *bias* es típicamente representado por la neurona *bias*, la cual siempre genera como salida el valor de 1.

Un perceptrón con dos entradas y tres salidas es representado en la Figura 3.3. Este perceptrón puede simultáneamente clasificar instancias en tres clases binarias diferentes, lo cual lo hace un clasificador multisalida.

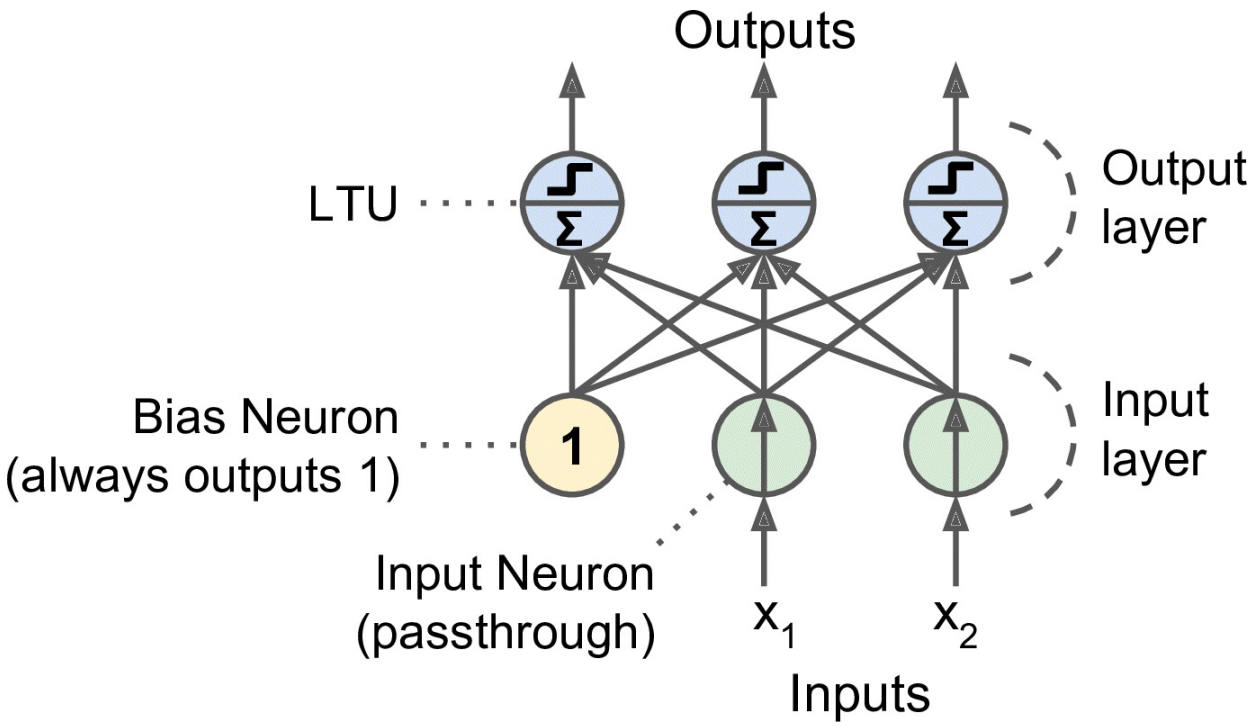


Figura 3.3: Diagrama del Perceptrón[Géron, 2017].

Para entrenar un perceptrón, se le ingresa a este una muestra a la vez, entonces el sistema generará la predicción para cada muestra. Por cada salida de una neurona que produjo una predicción equivocada, el sistema refuerza las conexiones con pesos que podrían haber contribuido a la predicción correcta.

3.1.4. Multi-Layer Perceptrón (MLP)

Un MLP esta compuesto por una capa de neuronas de paso o capa de entrada, una o más capas de LTUs, las cuales reciben el nombre de capas ocultas y una capa final de LTUs llamada capa de salida, como se aprecia en la Figura 3.4. Cada capa a excepción de la capa de salida incluye una neurona de *bias* y esta, está totalmente conectada a la próxima capa. Cuando una red neuronal artificial tiene dos o más capas ocultas esta recibe el nombre de *deep neural network* (DNN).

Para entrenar un MLP se emplea el algoritmo de *backpropagation* (retropropagación). Siguiendo este enfoque, para cada muestra de entrenamiento, el algoritmo la introduce a la red y calcula la salida de cada neurona en cada capa consecutiva (esto es llamado hacer el *forward* de la red, similar a cuando se hacen las predicciones). Luego se mide el error de salida de la red (la diferencia entre la salida desea u la salida real producida por la red) para luego calcular como cada neurona en la ultima capa oculta contribuyo al error de las neuronas de salida. Este es luego procesado para medir cuanto de este contribuciones al error vinieron de cada neurona de la capa anterior, y así

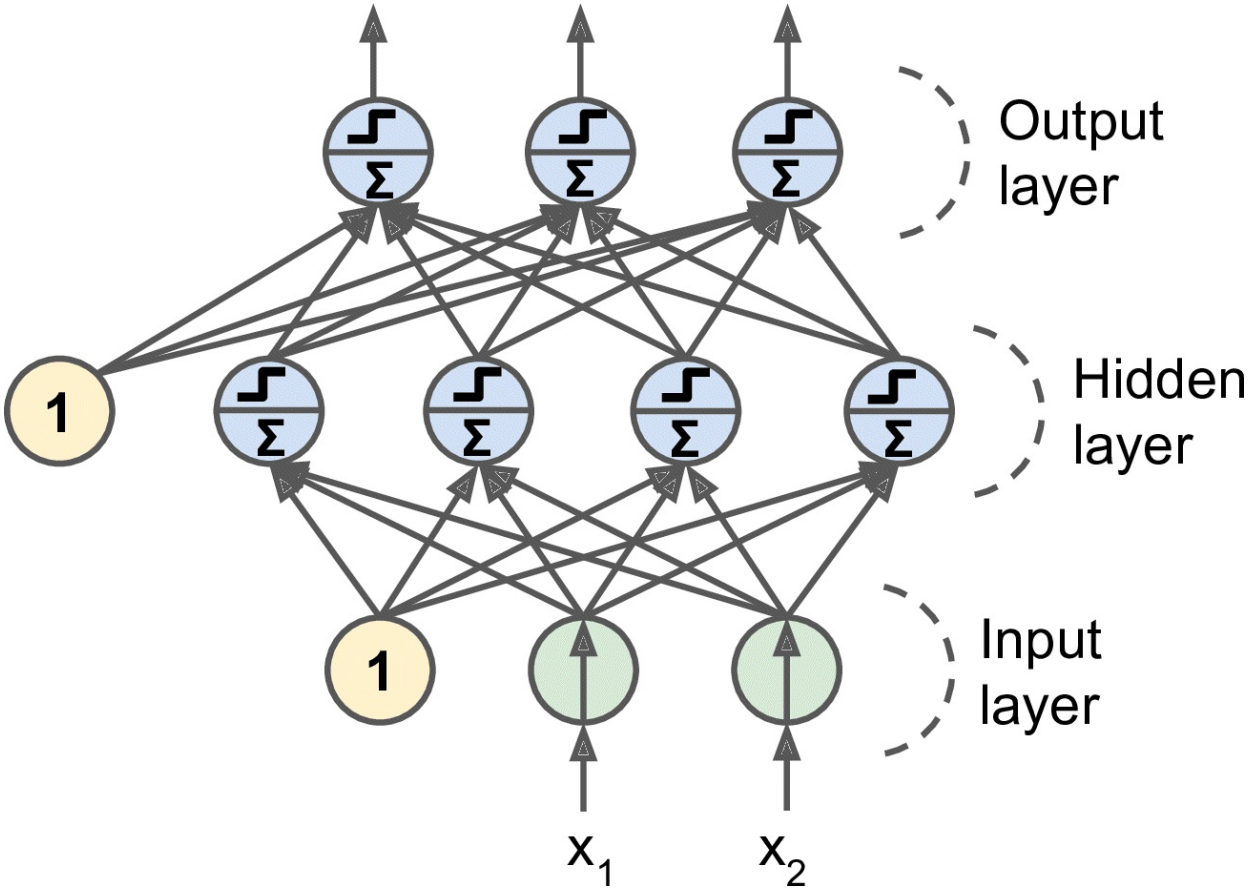


Figura 3.4: Multi-Layer Perceptron[Géron, 2017].

sucesivamente hasta que llega a la capa de entrada. Este paso en reversa mide eficientemente el gradiente del error a través de todos las conexiones de pesos en la red, mediante la propagación en reversa del gradiente del error.

En manera resumida se puede decir que: para cada instancia de entrenamiento el algoritmo de *backpropagation* primero realiza una predicción (hacer el *foward*), mide el error y entonces atraviesa cada capa en reversa para medir la contribución al error de cada conexión (paso en reversa) y al final retoca ligeramente las conexiones de pesos con la idea de reducir el error. Para que estos algoritmos funcionen, los autores de los mismos han reemplazado la función de paso del MLP por una función logística, ReLU o *tanh*. Esto se debe a que la función de paso solo contenía segmentos planos, por lo tanto no se tenían gradientes con los cuales trabajar. Mientras que la función logística tiene una derivada no cero bien definida todo el tiempo, lo cual permite al gradiente descendente hacer un progreso en cada paso.

Un MLP es usualmente utilizado para clasificación, con cada salida correspondiendo con una clase binaria diferente. Cuando las clases son exclusivas (clasificar numero del 0 al 9), la capa de salida es típicamente modificada mediante el reemplazo de la función individual de activación, por

una función compartida llamada *softmax* (Figura 3.5). Con esta función la salida de cada neurona corresponde con la probabilidad estimada de la clase correspondiente. En este tipo de redes la señal viaja solo en una dirección (de entrada a salida), por lo tanto la arquitectura es un ejemplo de *feedforward neural network* (FNN).

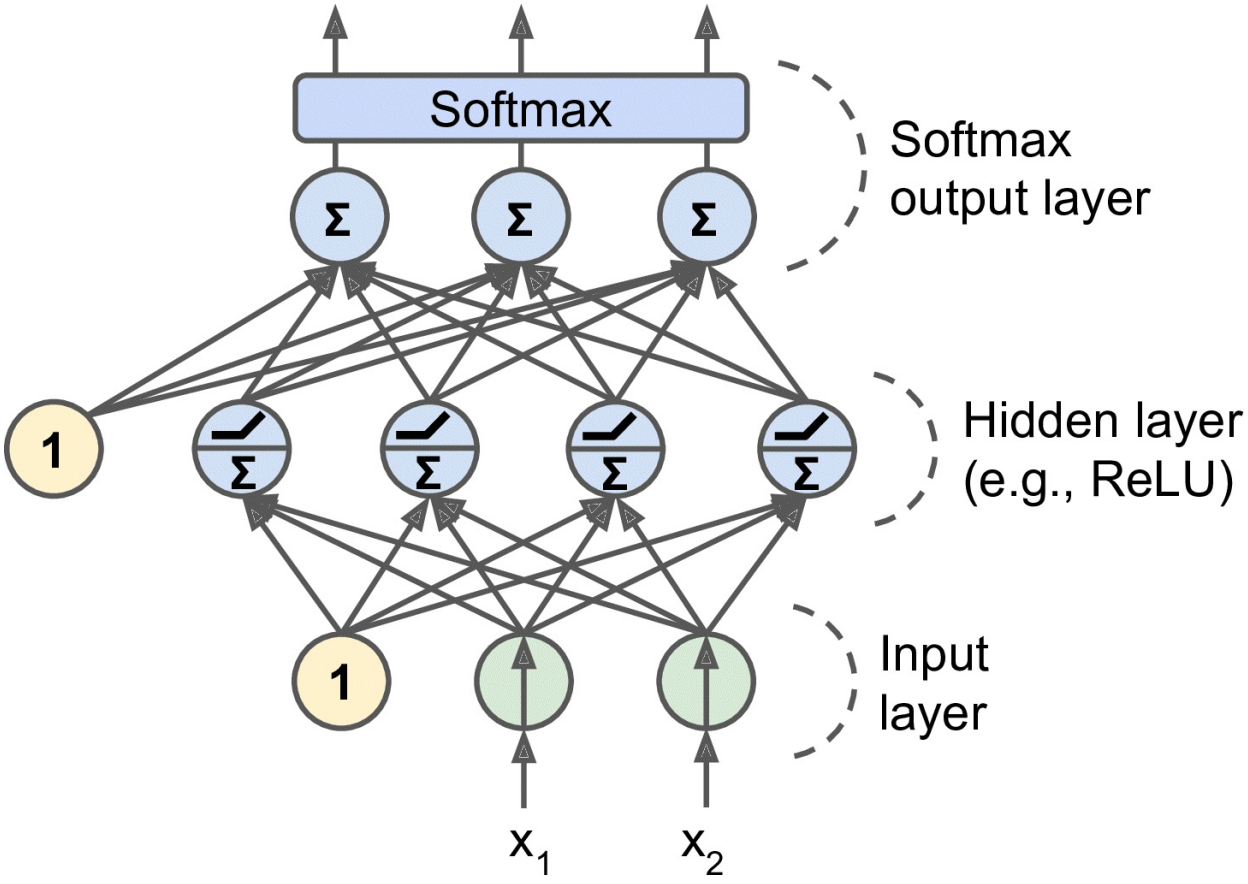


Figura 3.5: Un MLP moderno, con capas de ReLU y *Softmax*[Géron, 2017].

3.2. Redes Neuronales Convolucionales (CNNs)

A pesar de que Deep Blue el computador de IBM venció al campeón de ajedrez en 1997, hasta hace poco tiempo las computadoras aun eran incapaces de ejecutar ciertas tareas de manera confiable, las cuales son aparentemente fáciles para los humanos. Como por ejemplo detectar un cachorro en un fotografiá o reconocer las palabras en una frase hablada. Lo cual nos hace preguntarnos porque estas tareas son tan fáciles para los humanos? La respuesta esta en el hecho de que la percepción es un proceso que en gran parte se lleva a cabo fuera de los dominios de nuestra conciencia, con sensores y otros módulos especializados en visión, los cuales están el cerebro. Al momento que esta información llega a nuestra conciencia, esta ya esta adornada con características

de alto nivel; por ejemplo, al mirar la fotografiá del cachorro, no se puede elegir no ver al cachorro, o no notar su ternura. Tampoco se puede explicar como se ha reconocido a este tierno cachorro, esto es simplemente obvio para nosotros. En consecuencia, nosotros no podemos confiar en nuestra experiencia subjetiva: la percepción no es fácil y para comprenderla es necesario que se mire como funcionan los módulos sensoriales [Géron, 2017].

Las redes convolucionales (CNN del inglés, *Convolutional Neural Networks*) emergieron de un estudio de la corteza visual, y estas se han estado utilizando en reconocimiento de imágenes desde 1980. En los últimos años, gracias al incremento del poder computacional, a la gran cantidad de imágenes disponibles para entrenamiento y a diversos metodologías para entrenar redes profundas, las CNNs han alcanzado un desempaño super humano en algunas tareas visuales complejas. Estas han potenciado servicios de búsquedas de imágenes, autos autónomos, sistemas de clasificación automática de videos y muchos más. Ademas, CNNs no están restringidas a la percepción visual, sino que también están teniendo éxito en otras tareas como el reconocimiento de voz o el procesamiento de lenguaje natural.

3.2.1. Arquitectura del Cortex Visual

Entre 1958 y 1959 David Hubek y Torsten Wiesel llevaron acabo experimentos que dieron una idea del funcionamiento interno de la estructura del cortex visual(en gatos y luego en monos), por la cual recibierion un Nobel. Los estudios mostraron que muchas neuronas en el cortex visual tienen unos pequeños campos receptivos locales, significando que estas solo reaccionan a estímulos localizados en una región limitada del campo visual. Estos campos receptivos de diferentes neuronas, se solapan y en conjunto construyen el campo visual completo. Además, se demostró también que algunas neuronas solo reaccionan a imágenes de lineas horizontales, mientras que otros reaccionan a lineas con diferentes orientaciones. También se observo que algunas neuronas tenían campos receptivos más grandes, los cuales reaccionaban a patrones de combinaciones más complejos, formados a partir de patrones de bajo nivel. Estos resultados llevaron a la idea de que las neuronas de alto nivel se basan en las salidas de las neuronas cercanas de bajo nivel. Esta arquitectura es capaz de detectar todo tipo de combinaciones complejas en cualquier área del campo visual. En la Figura 3.6 se puede observar que cada neurona esta conectada solo a unas cuantas neuronas de la capa previa.

Estos estudios inspiraron el neocognitron en 1980, lo cual gradualmente evolucionó en lo que hoy se conoce como Redes Neuronales Convolucionales (CNNs). Un hito importante en este campo fue la presentación de la arquitectura LeNet-5 en 1998, por Yann LeCun et al. Dicha arquitectura es ampliamente utilizada en para reconocer números escritos a mano. Las CNN se componen de diversos tipos de bloques como capas totalmente conectadas, capas de funciones de activación, capas convolucionales y capas de pooling.

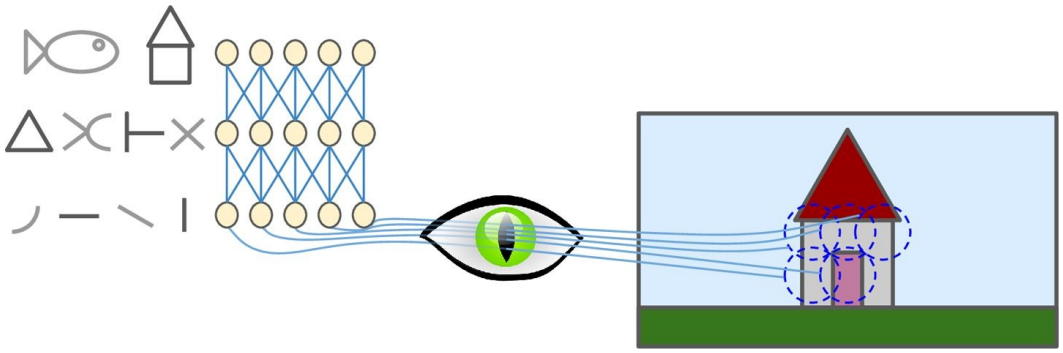


Figura 3.6: Campos receptivos locales de la corteza visual[Géron, 2017].

3.2.2. Definición

Las CNNs son definidas como modelos de aprendizaje de máquina jerárquicos, los cuales aprenden representaciones complejas de imágenes a partir de grandes volúmenes de datos anotados [Krizhevsky et al., 2012]. Estas utilizan múltiples capas de transformaciones básicas las cuales finalmente generan una representación sofisticada de alto nivel de la imagen [Clarifai, 2015]. En la Figura 3.7 se muestra la arquitectura básica de una CNN.

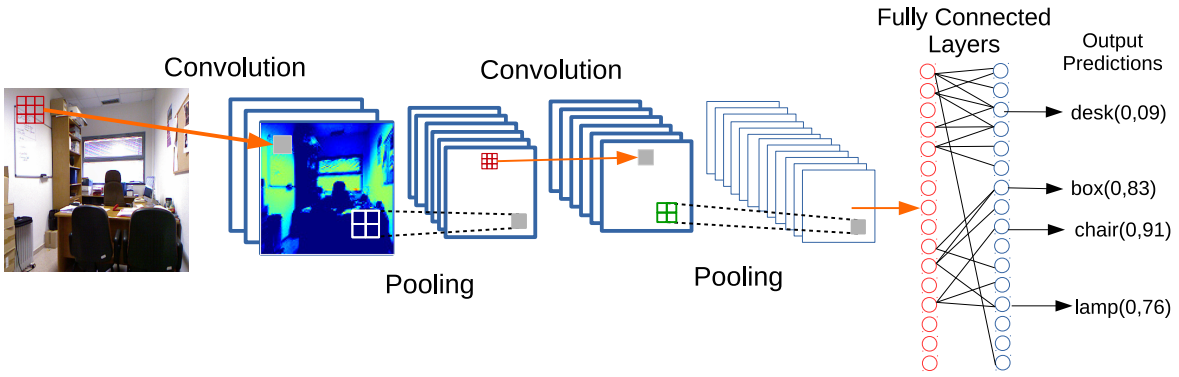


Figura 3.7: Arquitectura estándar de una CNN[Rangel, 2017].

Las CNN son una clase de red neuronal que se enfoca en el procesamiento de imágenes para obtener un conjunto de características, las cuales describan la imagen completamente. Esta descripción se logra mediante la concatenación de varios tipos de capas de procesamiento.

Estas redes pueden ser separadas en dos secciones diferentes las cuales podemos llamar: extracción de características y clasificación. Al inicio, la extracción de características se encarga del computo de las características de una imagen mediante la utilización de capas de convolución y *pooling*. Luego, la segunda sección, sección de clasificación, es responsable de calcular la salida de la red basándose en las características extraídas en la primera sección. Esta salida se calcula em-

pleando capas de neuronas totalmente conectadas al final de la red neuronal. El tamaño (cantidad de salidas) está determinado por la cantidad de categorías en las cuales la red ha sido entrenada para reconocer.

El uso de las CNNs en problemas de clasificación requiere pasos previos para lograr producir un clasificador. La primera fase es la recolección y ordenamiento de la información (imágenes en el caso de las CNNs) la cual será utilizada para entrenar un modelo, en otras palabras, la confección del conjunto de entrenamiento. Las imágenes recolectadas se agrupan en varias categorías atendiendo a las clases que la CNN debe reconocer. Entonces, la segunda fase corresponde con el proceso de entrenamiento. Esto es un paso crucial que definirá cómo de preciso es nuestro modelo de clasificación. La fase entrenamiento de una CNN funciona mediante el análisis del conjunto completo de entrenamiento. Cada imagen del conjunto pasará a través de las capas de la CNN para enseñarle a la red a emparejar las características extraídas con la clase correspondiente.

Las capas que se encuentran entre la primera capa (capa de entrada) y la última capa (capa de salida) son denominadas capas ocultas de la red neuronal. Estas varían en número según la arquitectura de la red y las características de las imágenes a procesar.

3.2.2.1. Capas Convolucionales

Usualmente la primera capa de una CNN es una capa convolucional. Este tipo de capas trabaja mediante la aplicación de filtros (*kernels*) (Figura 3.8) a la información de entrada. Estos filtros son aplicados a manera de ventana deslizante sobre la imagen de entrada. Consecuentemente, la imagen es parcialmente analizada mediante trozos de un tamaño definido. Los filtros, al igual que las imágenes son definidos mediante dos parámetros, altura y anchura, usualmente con igual valor. Actualmente se emplean tres canales para la información de color, pero inicialmente las imágenes de entrada fueron en escala de grises. Por consiguiente, el tamaño de un filtro se puede definir como: $5 \times 5 \times 3$ (Figura 3.9).

Por lo tanto, cuando un filtro se aplica sobre una porción de imagen, se lleva a cabo una multiplicación elemento a elemento y sus resultados son sumados para obtener la respuesta del *kernel* para la porción correspondiente de la imagen. Esta multiplicación (o aplicación de *kernel*) es repetida hasta que la imagen completa haya sido analizada. La salida del filtro produce un mapa de activación de los datos de entrada donde este mapa está compuesto del conjunto completo de respuestas del *kernel* para la imagen. Usualmente este mapa tiene un tamaño menor al de la imagen original, aunque otros enfoques se pueden utilizar para manejar los bordes de la imagen. Comúnmente los filtros incluyen un valor de desplazamiento el cual determina la cantidad de pixeles que el filtro se moverá en una dirección definida.

Los filtros se pueden definir como el campo receptivos de las neuronas en la red, de tal manera que cada neurona en la red no esta conectada a un píxel de la imagen, sino solo a su campo receptivo. De hecho, cada neurona en la segunda capa convolucional está conectada solo a las neuronas en un pequeño rectángulo de la capa previa. Es está característica la cual permite a la

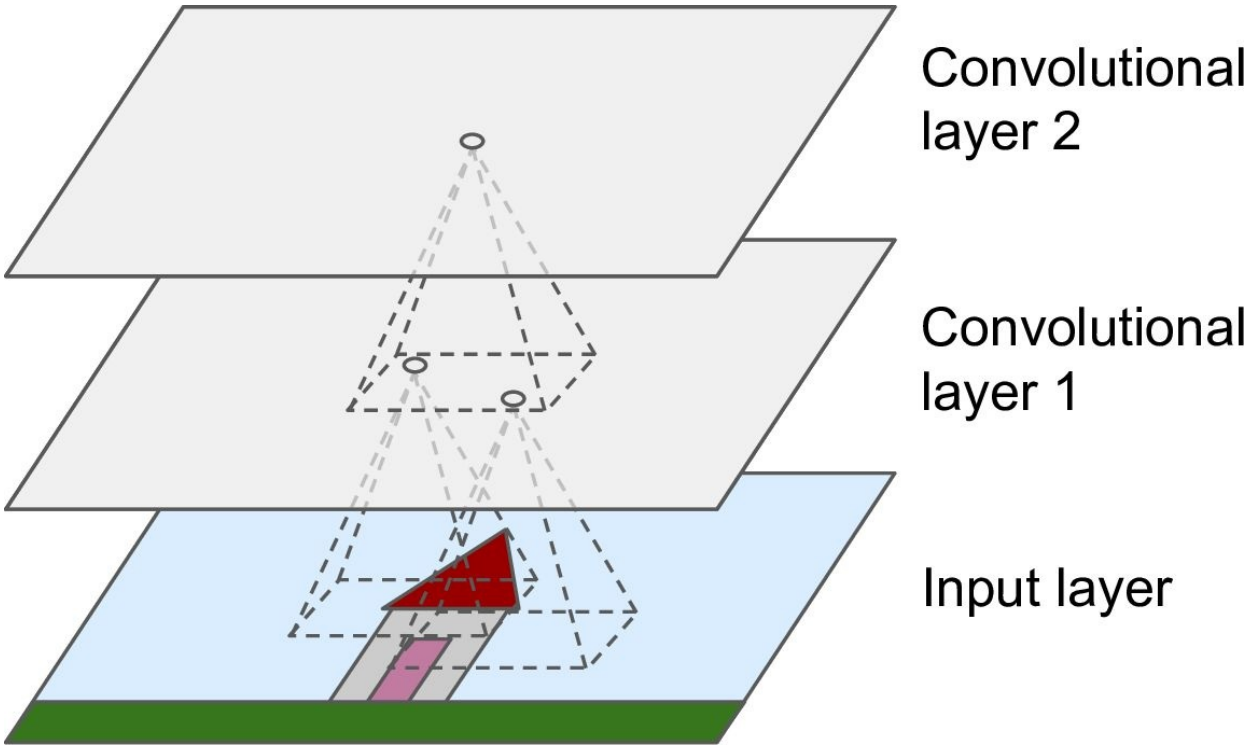


Figura 3.8: Capas de una CNN con sus filtros (campos receptivos)[Géron, 2017].

red concentrarse en características de bajo nivel en las primeras capas y luego ensamblarlas en características de alto nivel en las capas posteriores y así sucesivamente con todas las capas de la red. Esta estructura jerárquica es común en imágenes del mundo real, lo cual es una de las razones por las cuales las CNN funcionan muy bien para reconocimiento de imágenes.

Una capa convolucional puede tener varios tipos de filtros, por lo cual, la salida de una capa convolucional no es solamente un mapa de activación, sino un conjunto de estos, uno por cada *kernel* en la capa convolucional. En la Figura 3.11 se aprecian los diferentes mapas de activación que son producidos por cada capa de la red. La Figura 3.10 muestra el funcionamiento de un *kernel* de convolución. La aplicación de diferentes filtros a una sola imagen se muestra en la Figura 3.12.

Los *kernels* en las capas convolucionales, buscan la aparición de características específicas en la imagen de entrada. Estos *kernels* usualmente empiezan identificando características simples como puntos o curvas en la primera capa convolucional, luego las capas siguientes se especializan en identificar grupos de estas características y así sucesivamente encontrar características complejas y de más alto nivel como cajas o círculos.

3.2.2.2. Funciones de Activación de las Neuronas

Las Unidades de Rectificación Lineal o ReLUs (por sus siglas en inglés de *Rectifier Linear Units*) utilizan la salida de una capa convolucional y aplican la función de activación no saturada

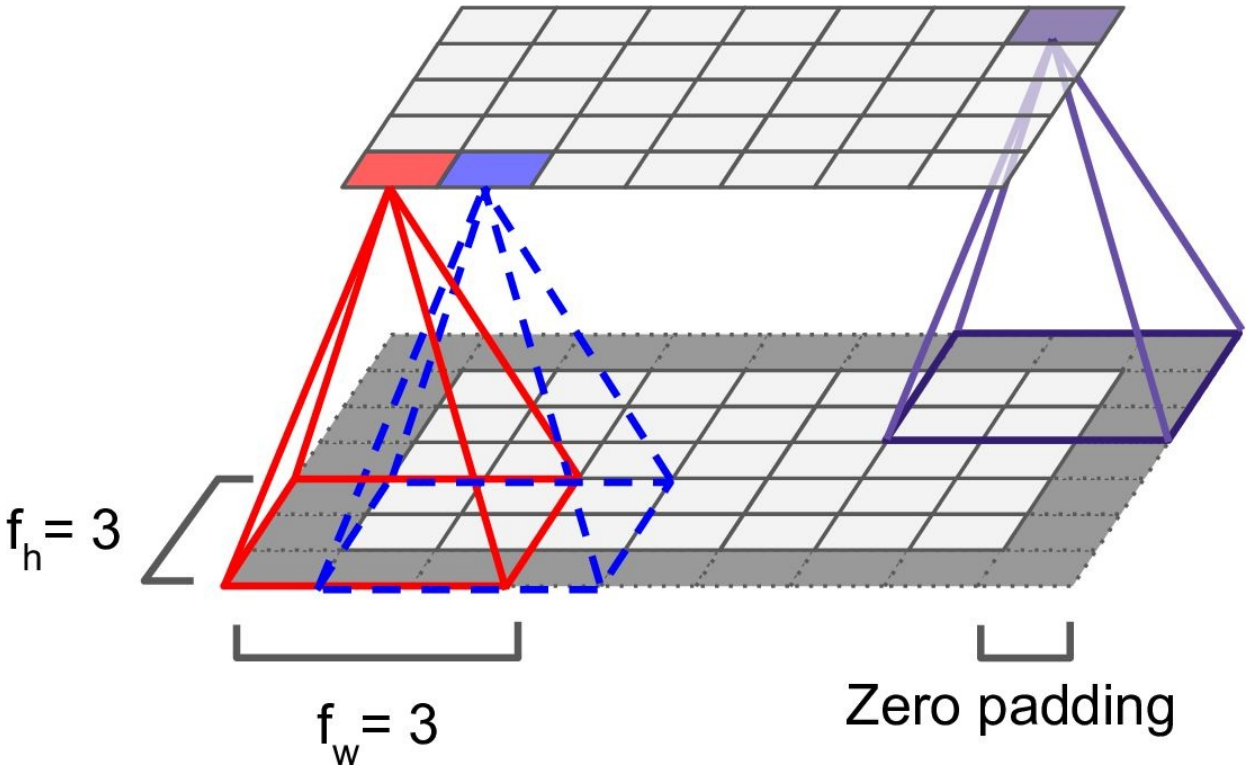


Figura 3.9: Tamaños de filtros y sus conexiones entre capas[Géron, 2017].

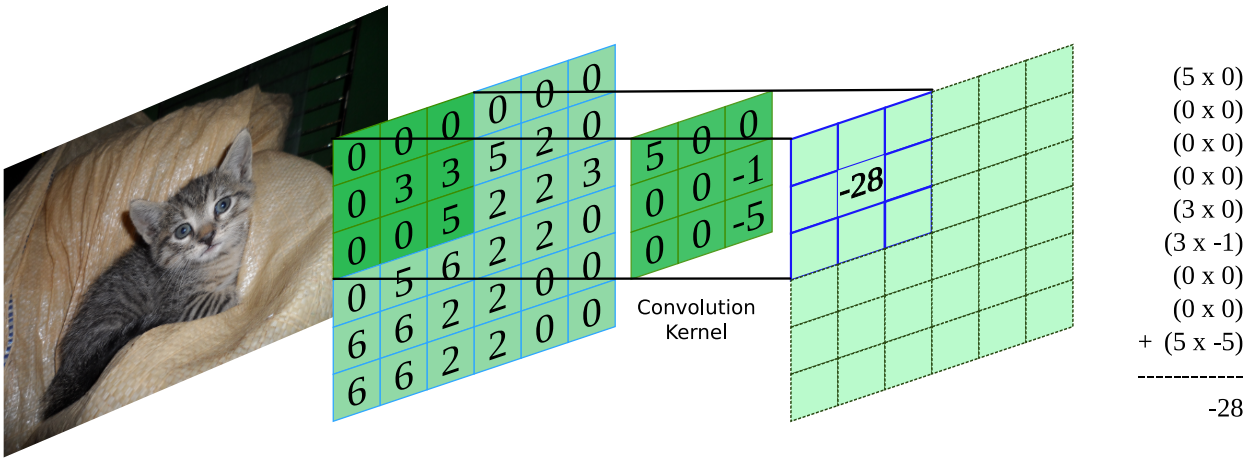


Figura 3.10: Ejemplo de convolución[Rangel, 2017].

$f(x) = max(0, x)$ (Figura 3.13 izquierda). La utilización de las ReLUs en procedimientos de entrenamiento mejora el mismo, reduciendo el tiempo requerido para entrenar un modelo y de la misma manera ayuda a resolver el problema del gradiente evanescente que se presenta en estas redes. El objetivo de esta función es la introducción de la no-linealidad al sistema, debido a que

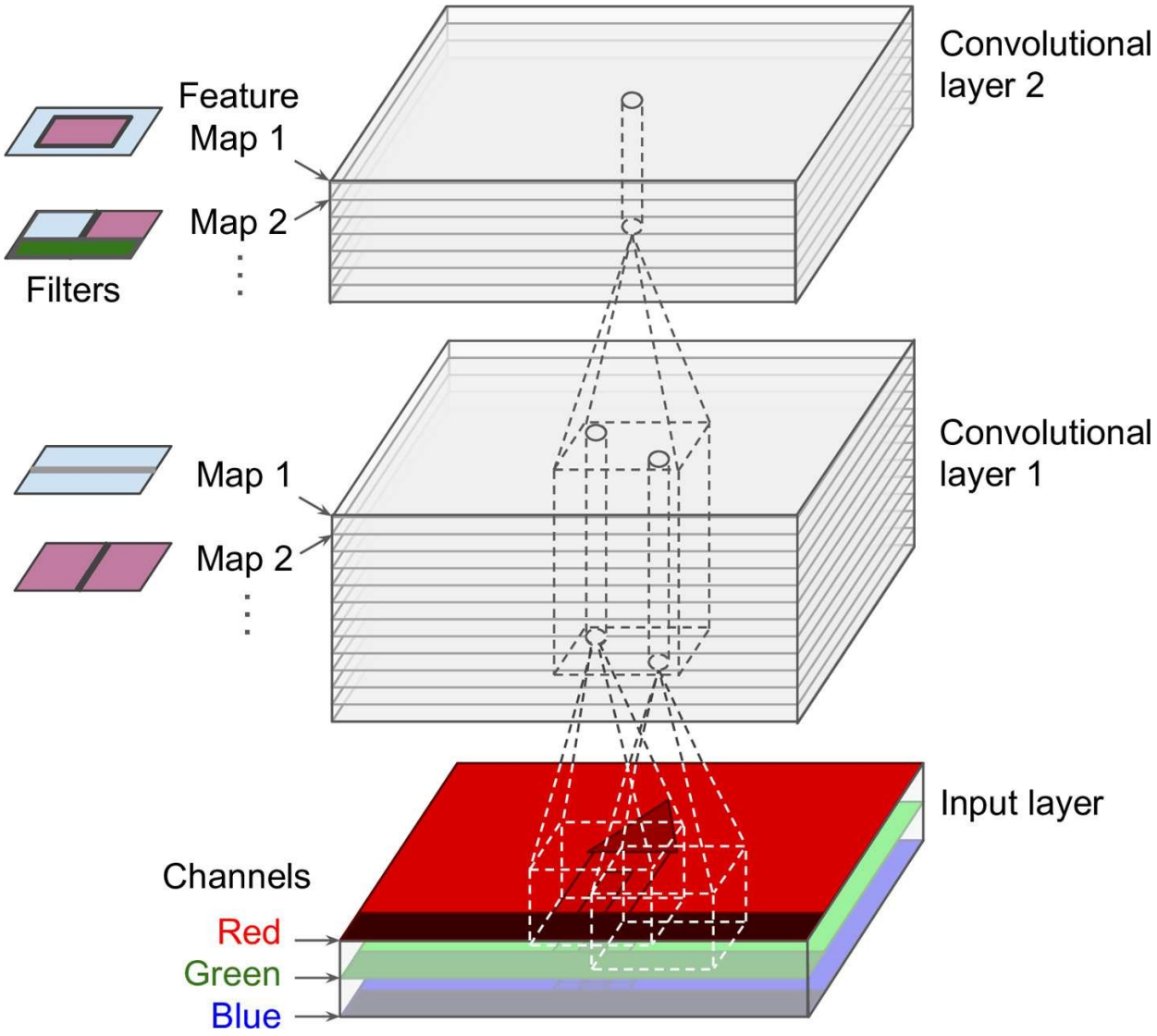


Figura 3.11: Capas de convolución con múltiples mapas de activación[Géron, 2017].

las capas convolucionales computan operaciones lineales (multiplicaciones elemento a elemento y sumatorios). Las funciones de activación ayudan a elevar las características no lineales del modelo y de la red. Este elevamiento se lleva a cabo sin modificar los campos respectivos de las capas convolucionales. En el pasado la modernización de la salida de una neurona estaba a cargo de funciones no lineales saturadas, especialmente la tangente hiperbólica $f(x) = \tanh(x)$ (Figura 3.13 centro), y la función sigmoidea $f(x) = (1 + e^{-x})^{-1}$ (Figura 3.13 derecha). Estas eran más lentas que las ReLUs, en términos de entrenamiento cuando se utilizaba la optimización mediante el algoritmo de gradiente descendente.

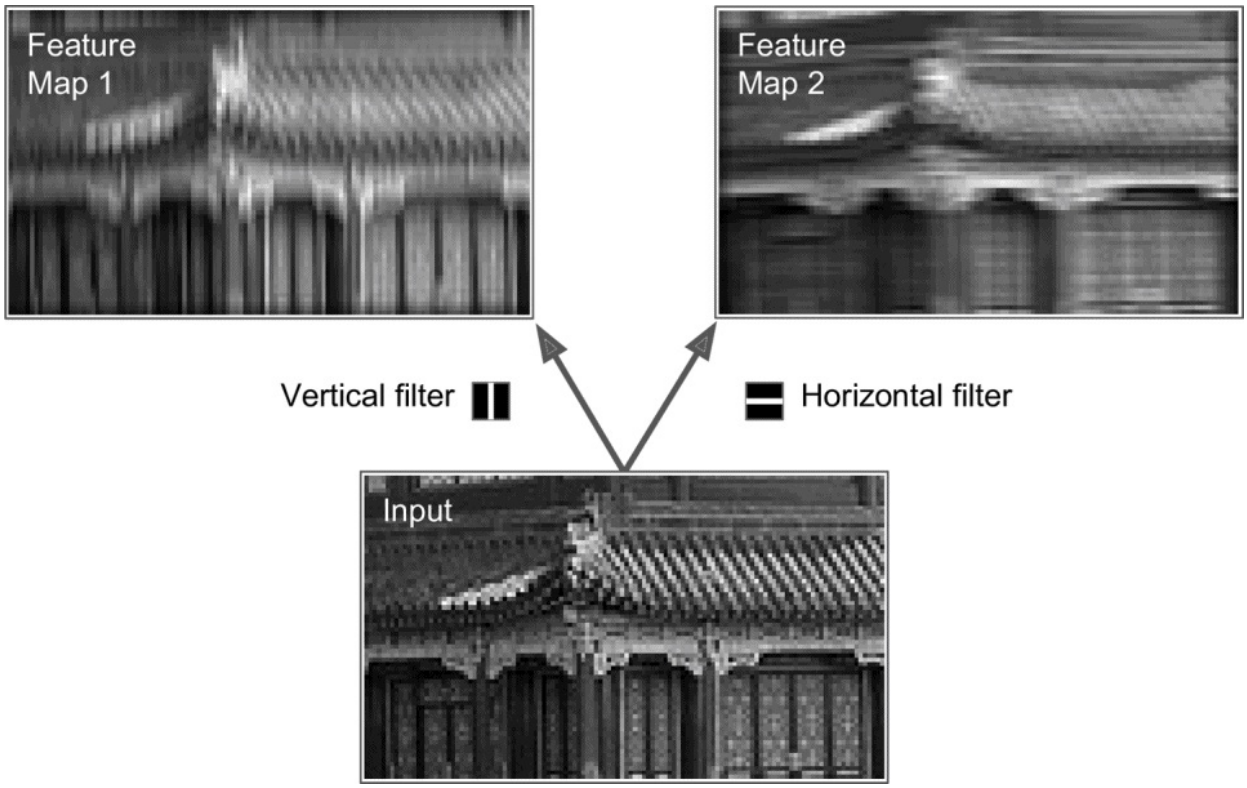


Figura 3.12: Aplicación de dos filtros diferentes para producir mapas de activación[Géron, 2017].

3.2.2.3. Capas de *Pooling*

Las capas de *pooling* actúan como un procedimiento de muestreo. Estas capas toman como entrada cada mapa de activación producida por una capa convolucional. De igual manera que las capas convolucionales, estas capas trabajan a manera de ventana deslizante con un tamaño definido. Por lo que, esta capa trabaja moviendo la ventana sobre la capa de activación, seleccionando un grupo de píxeles de la capa de activación y luego utilizando el valor máximo, mínimo o la media del grupo, reemplaza el grupo por dicho valor. Por lo cual, la salida de la capa de *pooling* es de menor tamaño que el del mapa de activación.

Esta secuencia de convolución y *pooling* es el esquema básico del esquema de trabajo de una CNN, pero no todo el tiempo una capa de *pooling* sigue a una capa de convolución. Algunas arquitecturas incluyen varias capas de convolución una después de otra. Es la arquitectura de la CNN la cual define la manera en que se dará la interacción entre las capas. La Figura 3.14 muestra un ejemplo de *max pooling* para uno de los canales de un mapa de activación de una capa convolucional.

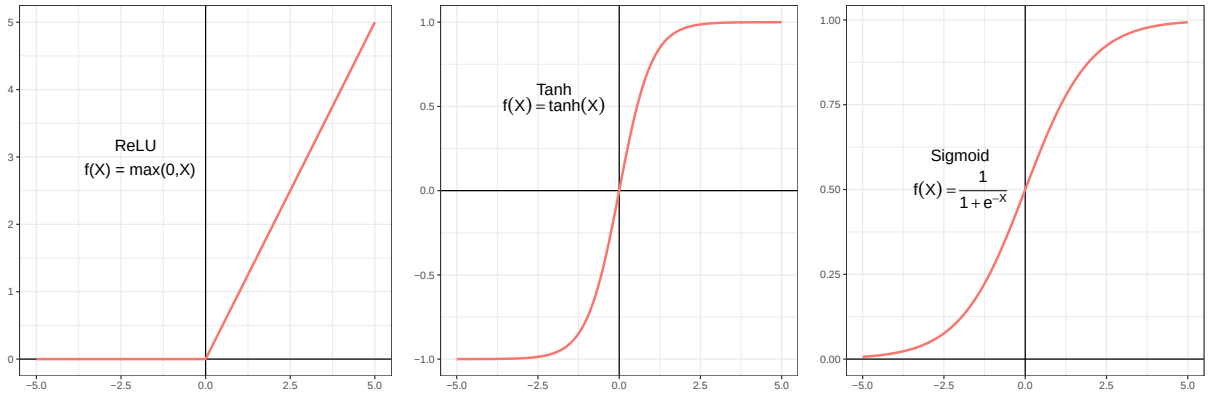


Figura 3.13: Funciones de Activación estándar

3.2.2.4. Capas Totalmente Conectadas

En una CNN las últimas capas corresponden a las capas totalmente conectadas las cuales se encargan de generar la salida de la red. Este tipo de capa trabaja de manera similar a un perceptrón multicapa (Figura 3.15), produciendo un vector n-dimensional como salida. El tamaño del vector dependerá de la cantidad de categorías entre la cuales la CNN es capaz de elegir. Por ejemplo, un clasificador del alfabeto solo producirá 26 salidas, una por cada letra.

Estas capas toman la salida producida por las capas anteriores (convoluciones, *pooling* o funciones de activación), y utilizan todos estos mapas de activación como su entrada. Por lo cual, cada mapa de activación está conectado a cada neurona de la capa totalmente conectada y multiplicada por un peso (ponderación) para producir la salida, determinando qué características de alto nivel están más correlacionadas con una categoría particular.

Básicamente las capas totalmente conectadas buscan correlaciones fuertes entre las características de alto nivel y una categoría particular con sus pesos particulares. Por lo tanto, una vez los productos entre las capas previas y los pesos se computen, las categorías obtendrán la probabilidad correcta.

Una capa totalmente conectada consiste de varios perceptrones multicapa. Estos se pueden describir como la función matemática que realiza un mapeado de un conjunto de valores de entrada (mapas de activación en CNNs) hacia un conjunto de valores de salida (categorías en las CNNs). Esta función está formada por la unión de muchas funciones simples. Podemos pensar que cada aplicación de una función matemática diferente como proveedora de una nueva representación de la entrada [Goodfellow et al., 2016]. En la Figura 3.16 se muestra una estructura usual de capas totalmente conectadas, donde tres de estas capas se concatenan para producir la salida de la red.

3.2.2.5. Capa *Softmax*

Las capas *softmax* toma la entrada generada por las capas totalmente conectadas y expresa estos valores como una distribución de probabilidad. Por lo cual, el sumatorio de los valores de

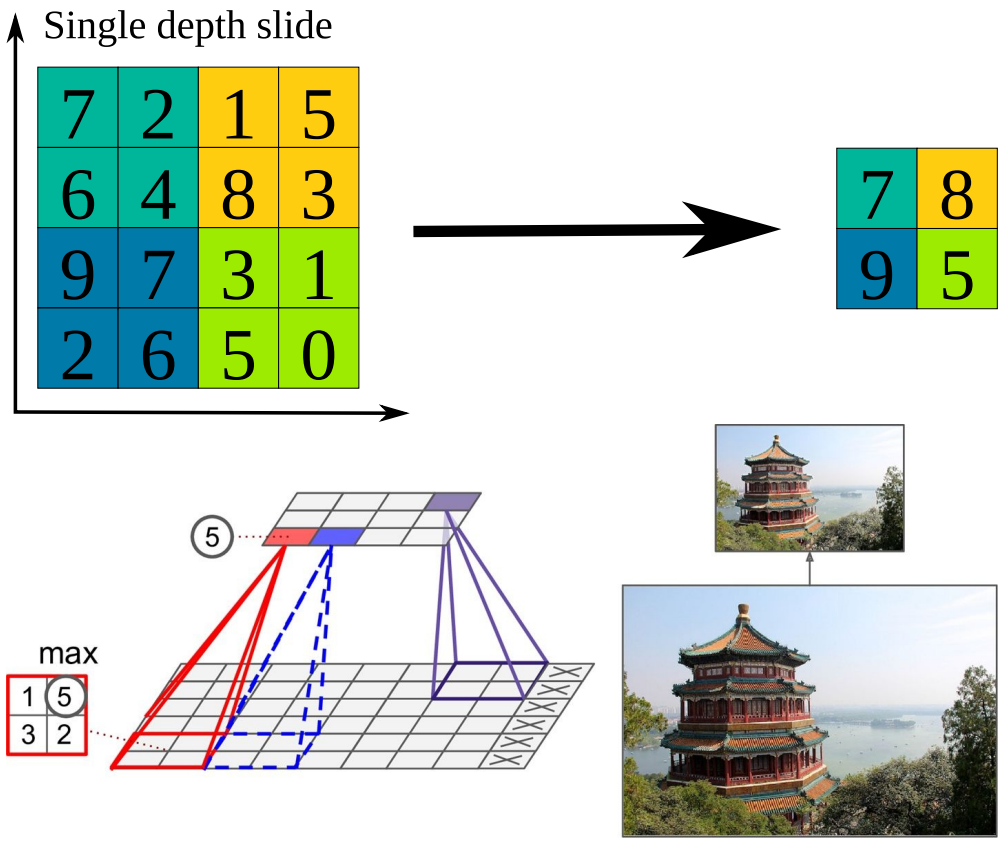


Figura 3.14: *Max Pooling*[Rangel, 2017][Géron, 2017].

salida de la red neuronal siempre sera igual a 1,0.

3.2.2.6. Retro-propagación (*Backpropagation*)

En las CNNs las diferentes capas ocultas producen una salida que será utilizada en la siguiente capa de la arquitectura. Estas salidas son utilizadas por las neuronas de las capas para producir su propia salida. La propagación de estos resultados entre las diferentes capas genera la respuesta de la CNN. Esta respuesta es producida por multiplicaciones sucesivas de los valores de las salidas y los pesos que representan la aportación de cada neurona a la respuesta final. Sin embargo, durante la fase de entrenamiento, el algoritmo de retro-propagación es el encargo de calcular cómo de bien se ajusta la salida de la CNN a la respuesta deseada, para cada instancia. Este algoritmo calcula un valor de error para cada salida. Luego, este valor es retro-propagado a las capas ocultas de la CNN, cada capa oculta se verá afectada por el valor de error dependiendo de cómo las neuronas de la capa aportaron a la respuesta calculada. Siguiendo la misma estrategia los pesos de los *kernels* en las capas convolucionales son actualizados. Este proceso asegura que las diferentes neuronas se puedan especializar en detectar diferentes características y así durante la etapa de prueba de la red, estas neuronas se activarán para una imagen nueva/desconocida [Goodfellow et al., 2016].

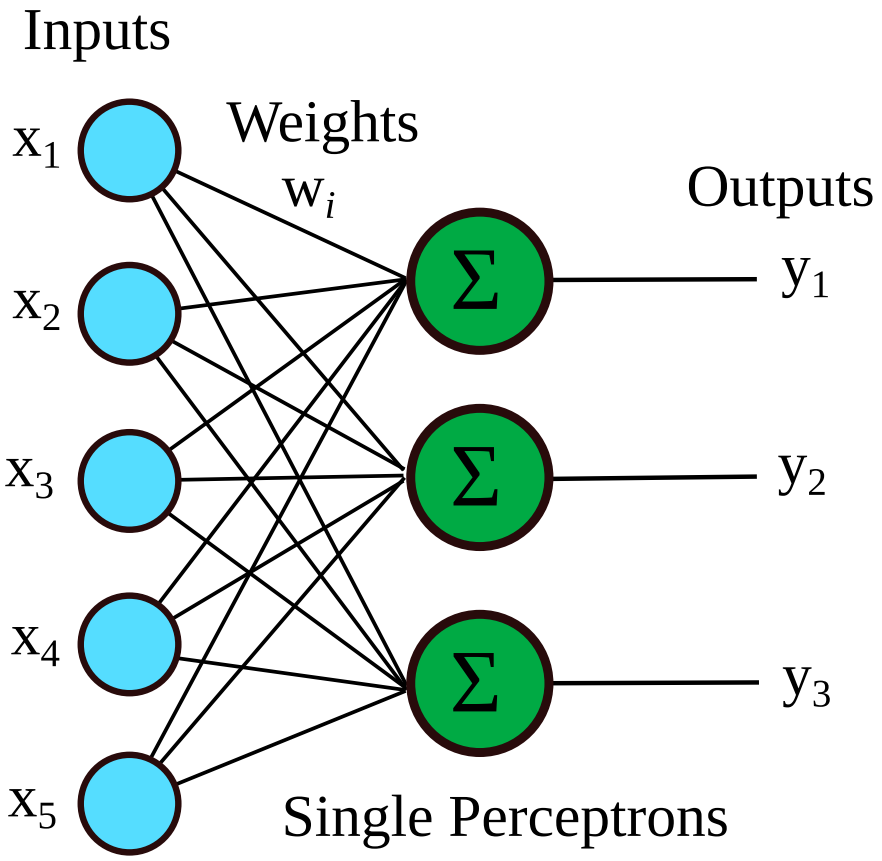


Figura 3.15: Perceptrón Multicapa[Rangel, 2017].

3.2.2.7. *Dropout*

Dropout [Srivastava et al., 2014] es un método de regularización para mejorar la ejecución de las redes neuronales mediante la reducción del sobre ajuste (*overfitting*). El algoritmo de retro-propagación puede llevar a desarrollar adaptaciones que funcionen para la secuencia de entrenamiento, pero sin la capacidad de generalizar para datos nunca vistos por el modelo. El método *Dropout* propone la desactivación aleatoria de las salidas de algunas neuronas, con el objetivo de romper estas adaptaciones haciendo independiente la presencia de cualquier unidad oculta particular. *Dropout* trabaja desactivando aleatoriamente unidades y sus conexiones en la red neuronal durante el proceso de entrenamiento de la red. La idea detrás de este algoritmo es evitar que las unidades se adapten demasiado a los datos de entrenamiento. [Goodfellow et al., 2016] define este algoritmo como la función que entrena el conjunto de todas las sub-redes que se pueden formar mediante la remoción de unidades (en las capas ocultas) de una red neuronal subyacente. Este proceso se puede realizar mediante la multiplicación de una salida de una neurona por un valor de cero.

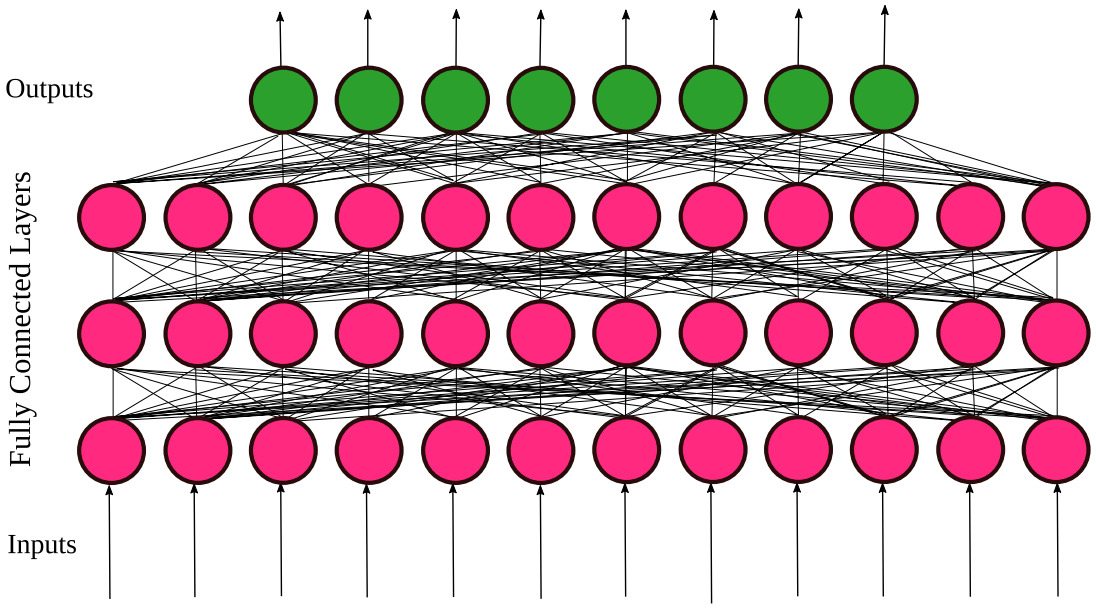


Figura 3.16: Capas totalmente conectadas[Rangel, 2017].

3.2.3. Arquitecturas de CNN

Cada CNN posee su propia arquitectura y esta se puede describir como la aplicación sucesiva de diferentes filtros y capas. Cada capa tiene la capacidad de reconocer características especificas de una imagen, desde un nivel bajo (píxeles) hasta patrones más detallados (formas). Cada CNN involucra la interacción de varias de estas capas. Según su función las capas pueden ser convolucionales, de *pooling* o totalmente conectadas, entre otras. Actualmente existen un amplio número de arquitecturas diferentes. Cada una tiene ciertas propiedades distintivas, como por ejemplo, la cantidad de capas convolucionales. GoogLeNet [Szegedy et al., 2014] y AlexNet [Krizhevsky et al., 2012] son dos arquitecturas ampliamente conocidas para su utilización en problemas de descripción/clasificación de imágenes en el campo de las CNNs.

Un modelo CNN puede ser definido como la combinación de un arquitectura y un conjunto de datos el cual fue utilizado como conjunto de entrenamiento para dicha arquitectura. La última capa de modelo de CNN es la responsable del proceso de clasificación y esta se encarga de realizar el mapeado de los valores calculados por las capas iniciales hacia la salida de la red la cual posee un tamaño definido. Cada arquitectura debe especificar el número de salidas que generará. El tamaño de esta salida corresponde a la dimensión del problema de clasificación. Cada modelo codifica la información recolectada del conjunto de datos seleccionado como conjunto de entrenamiento, por lo que, la salida de la red estará determinada por este conjunto.

Cada capa de una CNN produce modificaciones sobre la salida de la capa previa. En la Figura 3.17 se muestran las salidas producidas por algunas de la capas de una CNN, así como tambien, se visualizan las modificaciones producidas por cada capa. Al analizar la última capa de ReLU, es

distinguible como las CNNs son capaces de resaltar las regiones de la imagen donde están presentes los objetos.

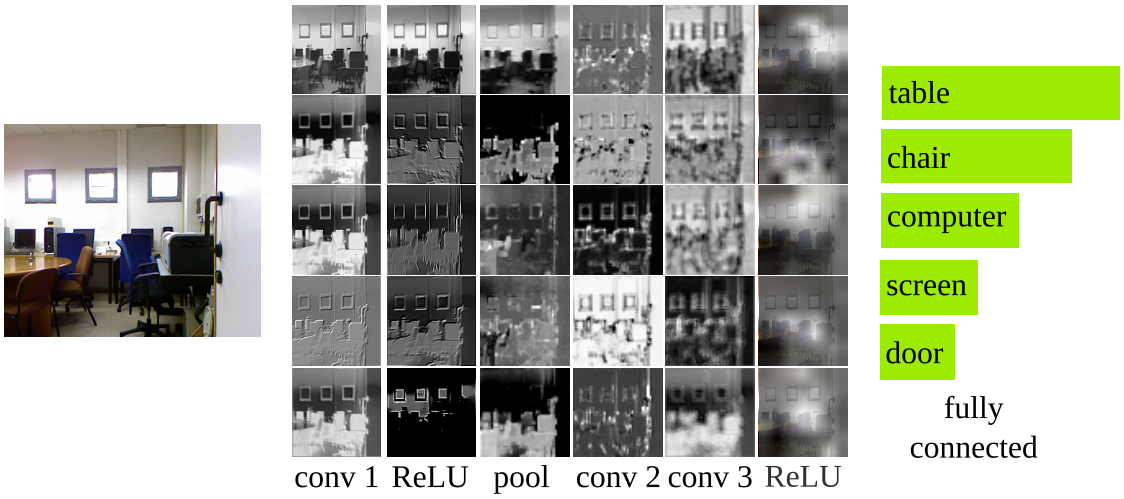


Figura 3.17: Resultados producidos por algunas de las capas de una CNN[Rangel, 2017].

3.2.4. Conjuntos de datos para los modelos Pre-entrenados de Caffe

3.2.4.1. ImageNet 2012

El conjunto de datos ImageNet 2012 [Russakovsky et al., 2015, Deng et al., 2009] es un subconjunto de ImageNet el cual está compuesto de más de 10 millones de imágenes pertenecientes a 21,000 categorías. Este subconjunto contiene más de un millón de imágenes de 1,000 categorías diferentes, por lo cual, para cada categoría este incluye aproximadamente 1,000 imágenes. ImageNet se creó para el reto de clasificación en el ImageNet edición 2012 y ha sido utilizado en la competencia durante los últimos años. Las categorías en el conjunto de datos incluyen objetos de diferentes tipos como: animales, instrumentos musicales y productos de aseo, entre otros. Este conjunto ha sido creado como una colección ordenada de imágenes representando objetos y escenas, jerárquicamente agrupados y ordenados por categorías. En la Tabla 3.1 se muestran imagénes de cuatro categorías en el conjunto de datos ImageNet.

3.2.4.2. *Places* 205

El conjunto de datos *Places* 205 [Zhou et al., 2014] consiste en una colección de 2,488,873 imágenes agrupados en 205 categorías de escenas. Este conjunto es un subconjunto del *Places* original el cual contiene más de siete millones de imágenes agrupadas en 476 categorías. *Places* 205 es un conjunto de datos centrado en escenas que se enfoca principalmente en categorías de interiores y exteriores. *Places* 205 fue creado por el MIT como una consecuencia de la falta de conjunto de datos centrados en las escenas para tareas de clasificación. Además, este conjunto fue

empleado para aprender características complejas de las imágenes utilizando varias arquitecturas de CNNs para clasificar imágenes. En la Tabla 3.2 se muestran imagénes de cuatro categorías en el conjunto de datos *Places* 205.

3.2.4.3. *Hybrid* MIT

El conjunto de datos *Hybrid* o Híbrido [Zhou et al., 2014] fue creado y presentando al mismo tiempo que *Places* y por los mismos autores. Este conjunto se construyó mezclando las imágenes pertenecientes a los ImageNet 2012 y *Places* 205, obteniendo 3.5 millones de imágenes. Por lo cual, se obtienen 1,183 categorías, donde las clases repetidas en los conjuntos originales fueron fusionadas. Este conjunto de datos fue originalmente construido para aprender características complejas empleando CNNs obteniendo buenos resultados en tareas de clasificación.

3.2.4.4. Modelos Pre-Entrenados con Caffe

Con el objetivo de entrenar un modelo CNN, debemos seleccionar una arquitectura y un conu para utilizar como conjunto de entrenamiento. La arquitectura específica los detalles internos como el número de capas de convolución o totalmente conectadas, así como también las operaciones espaciales utilizadas en las capas de *pooling*. Por otra parte, el conjunto de entrenamiento determina el número de etiquetas léxicas empleadas para definir una imagen.

Del conjunto completo de modelos pre-entrenados disponibles en el Caffe Model Zoo[1], hemos seleccionado siete diferentes candidatos los cuales se muestran en la Tabla **??**. Estos modelos difieren en sus arquitecturas, el conjunto de empleado para entrenamiento y el conjunto de etiquetas léxicas predefinidas para el modelo. Optamos por estos modelos debido a que todos están entrenados usando conjuntos de datos que están anotados con un gran número de etiquetas léxicas generalistas.

Caffe provee una forma sencilla de utilizar los modelos pre-entrenados liberados por su gran comunidad de usuarios. Tales modelos han sido entrenados con diferentes objetivos y son definidos por la combinación del conjunto de datos y la arquitectura empleada para su generación.

3.2.5. Arquitecturas CNN para los modelos pre-entrenados de Caffe

3.2.5.1. AlexNet

La arquitectura AlexNet [Krizhevsky et al., 2012] fue desarrollada para el reto ImageNet en su edición 2012, obteniendo el primer lugar en los resultados de la competición. Esta arquitectura sigue la definición básica de una CNN, utilizando una sucesiva combinación de capas de convolución y *pooling* y un conjunto de capas totalmente conectadas al final de diseño. La arquitectura de la red se compone de cinco capas convolucionales y tres capas totalmente conectadas. AlexNet modela

[1] `https://github.com/BVLC/caffe/wiki/Model-Zoo`

la salida de la red neuronal utilizando Unidades Rectificadoras Lineales (ReLUs) 3.2.2.2, reemplazando las funciones estándar *tanh*() o *sigmoid*. Las ReLUs mejoran la velocidad de entrenamiento de los modelos.

3.2.5.2. GoogLeNet

La arquitectura para CNN GoogLeNet [Szegedy et al., 2014] fue presentada para la competición ImageNet en su edición 2014, obteniendo el primer lugar en la competición y superando los resultados producidos por Clarifai en el 2013 y AlexNet en el 2012. La arquitectura incluye un nuevo concepto en el diseño de CNNs llamado módulo *inception* (Figura 3.18). Este módulo trabaja aplicando varios filtros de convolución de diferentes tamaños (*1x1, 3x3, 5x5*) a la misma entrada y en el mismo momento. Luego el resultado de cada filtro se concatena como la salida de la capa. Una capa incluye el procedimiento de *pooling* para la entrada. Este arreglo de pequeños filtros permite la utilización de menor cantidad de parámetros, lo cual mejora la ejecución del modelo. GoogLeNet está compuesto de 27 capas, de las cuales 22 tienen parámetros y las cinco restantes son capas de *pooling*. Sin embargo, el número de bloques individuales en el diseño es alrededor de 100.

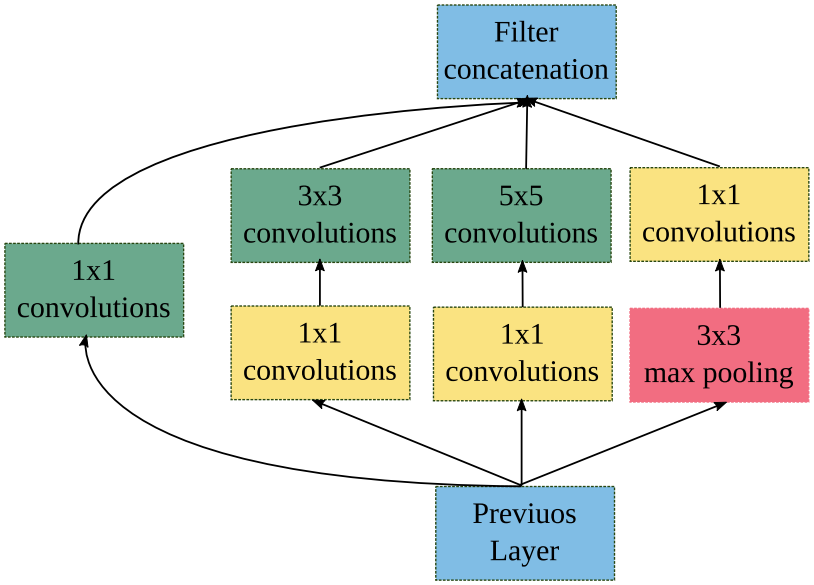


Figura 3.18: Módulo *Inception*.

3.3. Frameworks de Desarrollo

3.3.1. Caffe

Caffe, siglas de *Convolutional Architecture for Fast Feature Embedding* [Jia et al., 2014]. Este es un sistema rápido, modular y bien documentado, el cual es ampliamente utilizado por investigadores. Este entorno cuenta con una gran comunidad de usuarios que proveen modelos pre-entrenados que están listos para utilizar en cualquier versión de Caffe. La utilización de Caffe ha resultado en soluciones a diferentes tareas como son reconocimiento de objetos [Chatfield et al., 2014] o clasificación de escenas [Zhou et al., 2014]. Este sistema es desarrollado y mantenido por el centro de Visión y Aprendizaje de la Universidad de Berkeley (BVLC por sus siglas en inglés).

3.3.2. Tensor Flow

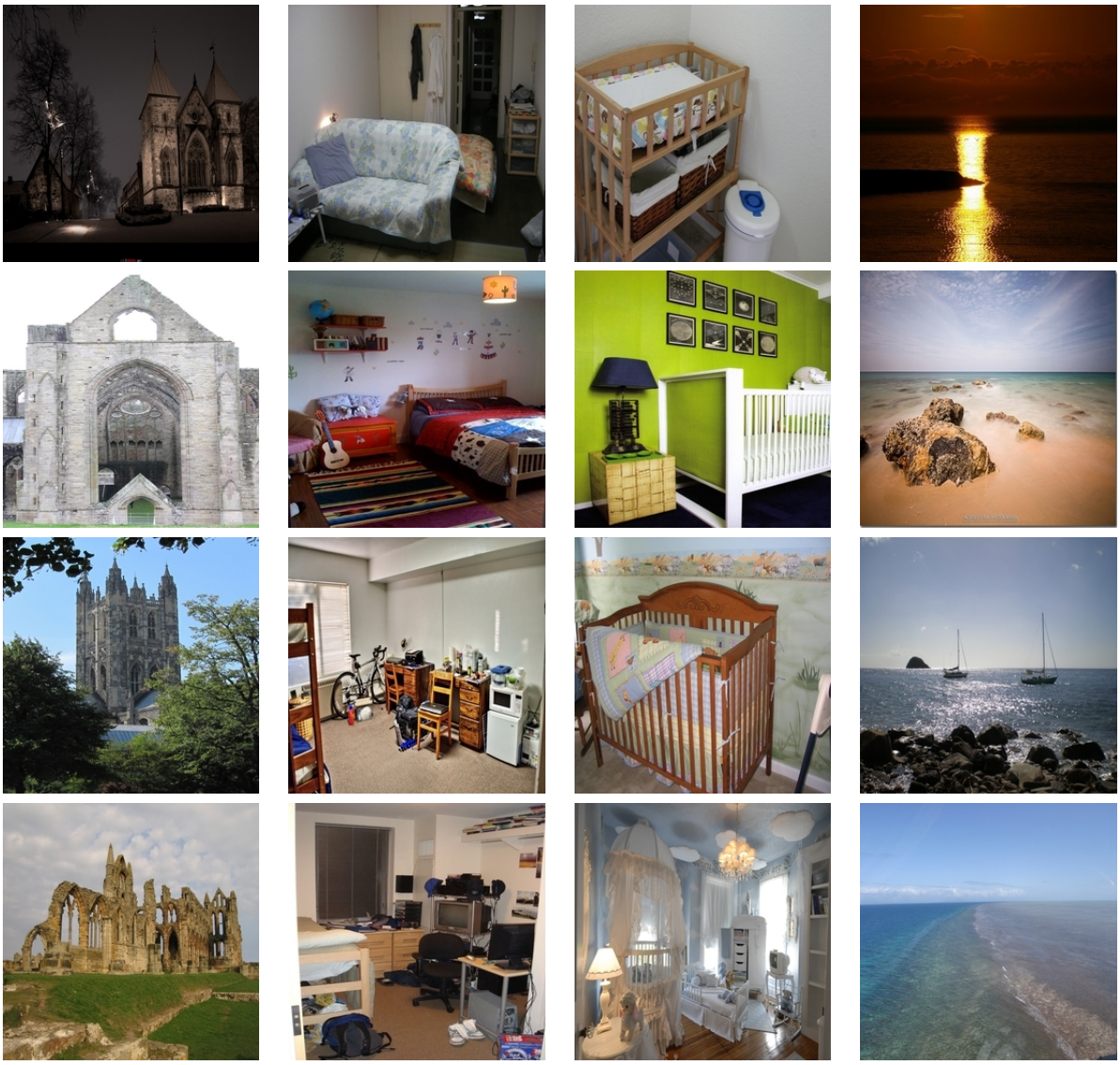
TensorFlow es una biblioteca de código abierto dirigida al aprendizaje automático a través de una serie de tareas. Ha sido desarrollado por Google para satisfacer las necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. Actualmente es utilizado tanto para la investigación como para la producción de productos de Google, remplazando el rol de su predecesor de código cerrado, DistBelief. TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015.

TensorFlow es el sistema de aprendizaje automático de segunda generación de Google Brain, liberado como software de código abierto a finales del año 2015. Mientras la implementación de referencia se ejecuta en dispositivos aislados, TensorFlow puede correr en múltiple CPUs y GPUs. Esta biblioteca también está disponible en Linux de 64 bits, macOS, y plataformas móviles que incluyen Android e iOS.

Cuadro 3.1: Imágenes de ImageNet

Taza	Martillo	Sax	Gato

Cuadro 3.2: Imágenes de *Places* 205



Abadía	Dormitorio	Dorm. Bebé	Océano

Capítulo 4

Redes Neuronales Recurrentes

4.1. Redes Neuronales Recurrentes RNN

Las redes neuronales recurrentes [Géron, 2017] constituyen una herramienta muy apropiada para modelar series temporales. Se trata de un tipo de redes con una arquitectura que implementa una cierta memoria y, por lo tanto, un sentido temporal. Esto se consigue implementando algunas neuronas que reciben como entrada la salida de una de las capas e inyectan su salida en una de las capas de un nivel anterior a ella.

En un juego de baseball, cuando un bateador golpea la pelota, un jardinero del equipo contrario inmediatamente empieza a correr, anticipando la trayectoria de la bola. La sigue y adapta sus movimientos y finalmente la atrapa. Durante este tiempo todo lo que se ha hecho es predecir el futuro, ya sea si se termine la frase que ha empezado un amigo o si se ha anticipado el olor una taza con café en el desayuno. Esta sesión trata de las Redes Neuronales Recurrentes (Recurrent Neural Networks, RNN, por sus siglas en inglés). Son un tipo de red que puede predecir el futuro (hasta cierto punto). Estas analizan datos de series de tiempo como el precio de las acciones e informar cuando comprar o vender. En sistemas de conducción autónoma, estas redes pueden anticipar la trayectoria del vehículo y así evitar accidentes. Mas generalmente, estas trabajan en secuencias de longitud arbitraria, al contrario de otras redes que reciben una entrada de tamaño fijo. Estas pueden tomar como entrada frases, documentos, muestras de audio; lo cual las hace extremadamente útiles para el sistemas de procesamiento de lenguaje natural, traducción automática, transcripción habla-a-texto o el análisis de sentimientos (por ejemplo leer las opiniones de una película y extraer el sentimiento general acerca de esta).

Ademas la habilidad de las RNN para anticipar, les da la capacidad de una creatividad sorprendente. Se les puede preguntar acerca de cuales son las siguiente notas más probable en una melodía. Luego que seleccione aleatoriamente una de estas y la toque. Repitiendo este proceso una y otra vez, la red ha sido capaz de crear una melodía tal y como lo hace el Proyecto Magenta de Google. De igual manera las RNNs pueden generar frases, descripciones de imágenes y mucho más. Los resultados no seran exactamente Mozart o Shakespeare, todavía, pero quien sabe que se pueda producir en unos pocos años a partir de ahora.

4.1.1. Neuronas Recurrentes

Las CNNs tienen un flujo de activación en una única dirección, desde la capa de entrada hasta la capa de salida. Una RNN se parece un poco a este tipo de red neuronal excepto en que tiene conexiones que están apuntando hacia atrás. Miremos a la posible RNN más simple, la cual se compone solo de una neurona recibiendo entradas, produciendo una salida y re-enviando esa salida a si misma. En cada instante de tiempo t (llamado frame), esta neurona recurrente recibe las entradas x_t así como también su propia salida del frame anterior, y_{t-1}. Esta pequeña red se puede representar en el eje del tiempo, como se muestra en la Figura 4.1, a lo cual se le conoce como desenrollar la red a través del tiempo.

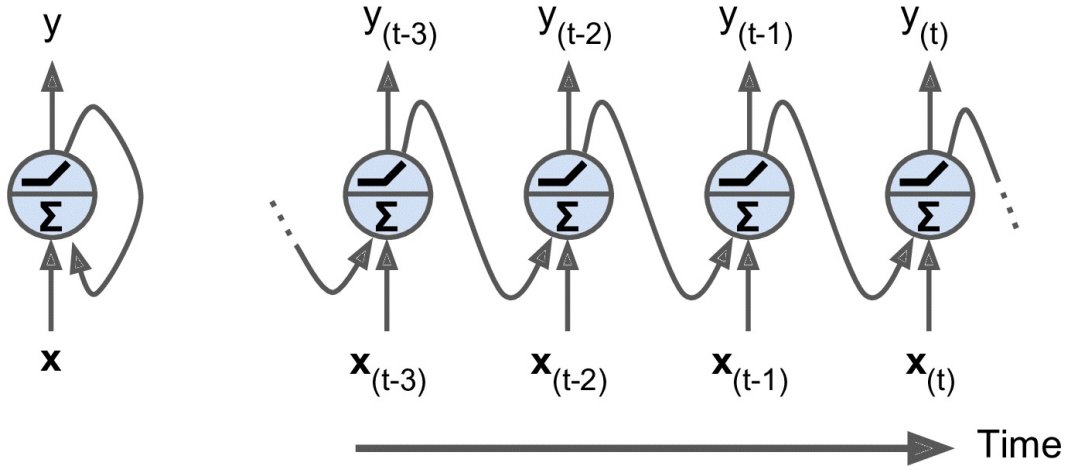


Figura 4.1: Una neurona recurrente (izq.), Una red desenrollada en el tiempo (der.)[Géron, 2017].

Se puede crear fácilmente una capa de neuronas recurrentes mostradas en la Figura 4.2. En cada tiempo t, cada neurona recibe el vector de entrada x_t y el vector de salida del paso anterior y_{t-1}. En este caso ambos (entrada y salida) serian vectores. De igual manera cada entrada y salida tendrá relacionados un vector de pesos, vectores w_x y w_y.

4.1.2. Células de Memoria

Debido a que la salida de una neurona recurrente en el tiempo t es una función de todas las entradas de tiempos previos, se puede decir que esta tiene una forma de memoria. La parte de la red neuronal que preserva algún estado a través del tiempo es llamada célula de memoria o simplemente celula. Una neurona simple o una capa de neuronas, es una célula muy básica, pero existen otras más complejas y poderosas.

En general el estado de una celda en el tiempo t se denota como h_t (la h viene de "hidden") es una función de algunas entradas en un instante de tiempo y su estado en el instante de tiempo previo: $h_t = f(h_{t-1}, x_t)$. Sus salidas en el instante de tiempo t denotado por y_t, es también una

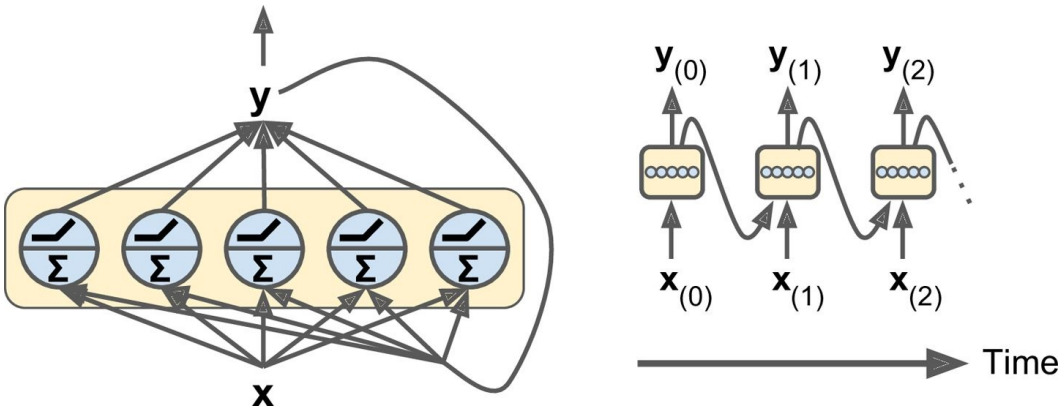


Figura 4.2: Una capa de neuronas recurrentes (izq.), Una red desenrollada en el tiempo (der.)[Géron, 2017].

función de los estados previos y las salidas actuales. En el caso de celulas básicas la salida es igual a estado, en en células complejas esto no siempre es así como se aprecia en la Figura 4.3.

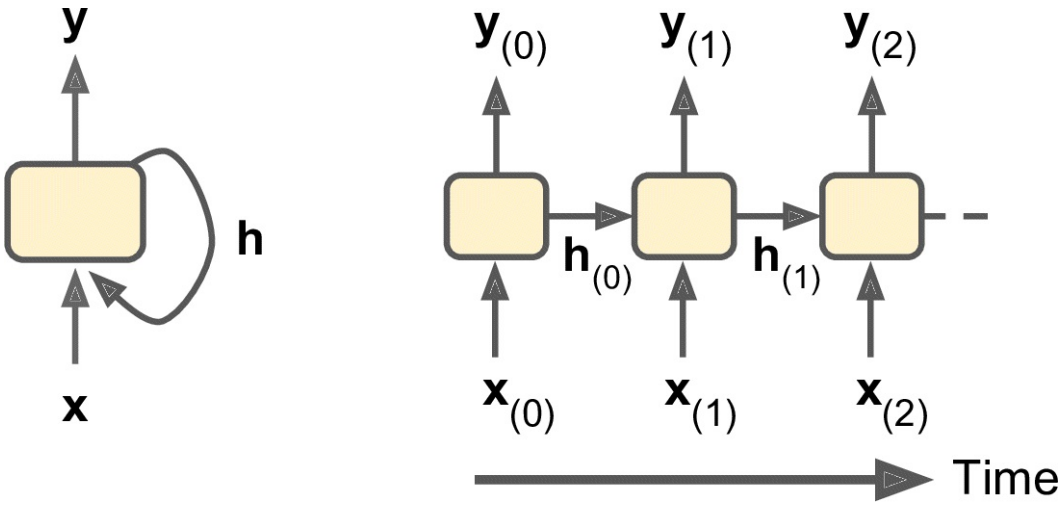


Figura 4.3: El estado oculto de la celula y su salida pueden ser diferentes[Géron, 2017].

4.2. Secuencias de Entrada y Salida

Una RNN puede simultáneamente tomar una secuencia de entrada y producir una secuencia de salidas, Por ejemplo, este tipo de red es útil para predecir series temporales como el precio de las acciones, esta tomara como entrada los precios en los N días y debe producir los precios de manera desplazada en un día en el futuro, es decir desde N-1 día atrás a mañana. Alternativamente también se puede enviar a la red una secuencia de entradas e ignorar todas las salidas con excepción de la

ultima. En otras palabras seria un red de Secuencia-a-Vector, por ejemplo se puede introducir en la red una secuencia que corresponde a comentarios sobre una película y la red producirá como salida una puntuación referente al sentimiento reflejado en los comentarios (desde -1 para odio hasta 1 para agrado).

De igual manera a la red se le puede introducir una entrada simple en el primer instante de tiempo (los demás instantes se rellenaran con 0s) y esta producirá una secuencia. Lo cual correspondería con una red Vector-a-Secuencia. Como ejemplo podemos tener como entrada una sola imagen y la salida seria la descripción de la misma.

Como otra alternativa de entrada se pueden tener una red Secuencia-a-Vector, llamada encoder, seguida de una red Vector-a-Secuencia, llamada decoder. Esta puede ser usada para traducir una frase de un idioma a otro. Se introduciría a la red una frase en un idioma, la primer parte de la red la procesaría y produciría una representación vectorial simple. Luego el decoder decodificaría este vector en una frase en un segundo idioma. Este modelo de 2-pasos, llamado Encoder-Decoder, trabaja mucho mejor que tratar de traducir sobre la marcha con una RNN de Secuencia-Secuencia, debido a que las 2 ultimas palabras de una frase pueden afectar las primeras palabras de la traducción, por lo tanto se requiere esperar hasta que se tenga la información completa de la frase antes de traducirla.

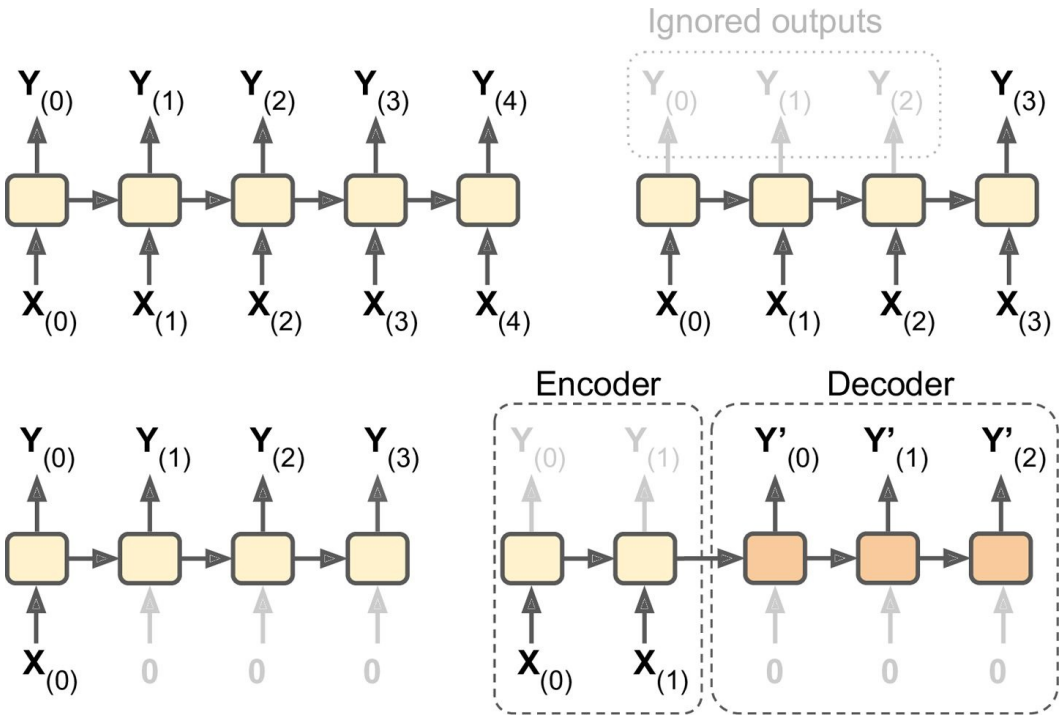


Figura 4.4: Secuencia-a-Secuencia (arriba izq.), Secuencia-a-Vector(arriba der.), Vector-a-Secuencia (abajo izq.), Secuencia-a-Secuencia desfasada(abajo der.)[Géron, 2017]

4.3. Célula Long Short-Term Memory

LSTM, fue propuesta por primera vez en 1997 por Sepp Hochreiter y Jürgen Schmidhuber y esta fue gradualmente mejorada con el pasar de los años por muchos otros investigadores. Si se considera la LSTM como una caja negra, esta puede ser utilizada de manera muy parecida a una célula básica, con la excepción de que esta se ejecuta mejor y que convergerá mucho más rápido y detectará dependencias a corto plazo en los datos. Las LSTM manejan 2 vectores de estado y por razones de desempeño estos se mantienen separados por defecto.

Si se observa el diagrama de una LSTM sin mirar sus interior se aprecia que esta luce exactamente como una celular regular, con la diferencia de que su estado es dividido en 2 vectores: h_t y c_t. Se puede pensar en h_t como el estado a corto plazo y c_t como el estado a largo plazo (Figura 4.5).

La idea principal es que la red puede aprender que almacenar en el estado a largo plazo, que descartar y que leer de este. A medida que el estado a largo plazo c_t atraviesa la red de izquierda a derecha, se puede observar que este va a través de una compuerta de olvido (*forget gate*) olvidando algunas memorias, y entonces esta añade algunas nuevas mediante la operación de adición. Esta operación añade memorias que fueron seleccionadas por una puerta de entrada (*input gate*). El resultado c_t se enviá directo hacia afuera de la celda, sin ninguna transformación. Así, en cada instante de tiempo, algunas memorias se descartan y otras se añaden. Además, después de la operación de añadido, el estado a largo plazo (c_t) es copiado y enviado a una función $tanh$ y luego es filtrado por la puerta de salida. Esto produce el estado a corto plazo h_t (el cual es igual para a la salida de la célula para este instante de tiempo y_t)

En resumen, una LSTM puede aprender a reconocer una entrada importante (trabajo de la *input gate*), almacenarlo en el estado a largo plazo, aprender a preservarlo mientras lo necesite (trabajo de la *forget gate*) y aprender a extraer lo que sea que necesite. Esto explica porque estas han sido tan exitosas capturando patrones a largo plazo en series temporales, textos largos, grabaciones de audio y más.

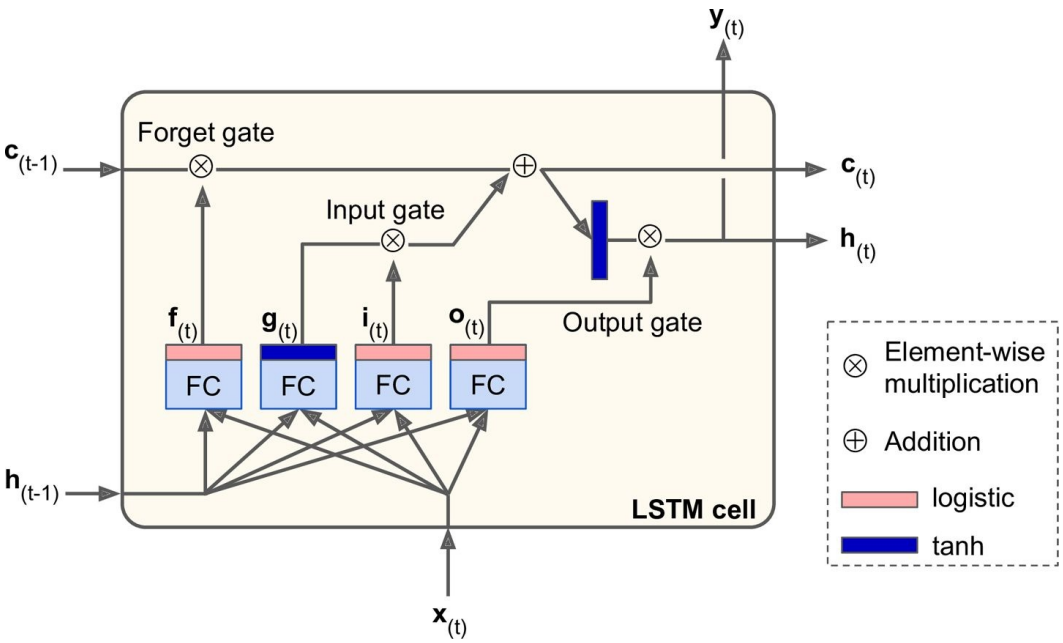


Figura 4.5: Celula *Long short-Term Memory*[Géron, 2017].

Capítulo 5

Generative Adversarial networks GAN

5.1. *Generative Adversarial networks GAN*

Las GANs, son una arquitectura neuronal profunda que esta compuesta de 2 redes, compitiendo una contra otra. Las GANs fueron introducidas por Ian Goodfellow [Goodfellow et al., 2014] y otros en el 2014. La Figura 5.1 muestra la red generativa (parte superior) y la red discriminativa (parte inferior) de una GAN.

El potencial de las GANs es inmenso debido a que ellas pueden aprender a imitiar cualquier distribucion de datos. GANs pueden aprender a crear mundos misteriosamente similares al nuestro en cualquier dominio: musica, imagenes, habla, texto entre otras. Estas son robots artistas en un sentido y sus salidas son impresionantes.

Para entender las GANs se debe conocer como trabajan los algoritmos generativos, para lo cual contrastarlos con los algoritmos discriminativos puede ser útil. Un algoritmo discriminativo trata de clasificar los datos de entrada. Esto es dados las características de una instancia de datos, el algoritmo trata de predecir una etiqueta o categoría a la cual la instancia pertenece.

Por ejemplo, dadas todas las palabras de un e-mail, un algoritmo discriminativo puede predecir si el mensaje es un *spam* o no. *Spam* es una de las etiquetas, y las palabras recolectadas en el correo, son las características que constituyen los datos de entrada.

Por lo tanto un algoritmo discriminativo hace un mapeado de características a etiquetas. Estos están unicamente interesados en esta relación. Una manera de pensar acerca de lo algoritmos generativos es que estos hacen lo opuesto. En lugar de predecir una etiqueta, dadas ciertas características, estos algoritmos tratan de predecir las características dada una etiqueta.

La pregunta que un algoritmo generativo trata de responder es dada una etiqueta, que tan probable es que estas características generadas pertenezcan a dicha clase. Mientras que un modelo discriminativo se enfoca más en la relación y y x. Los modelos generativos se preocupan por como obtener x. Un modelo discriminativo es capaz de aprender sobre el limite de las clases. Los modelo generativos moldean la distribución de clases individuales.

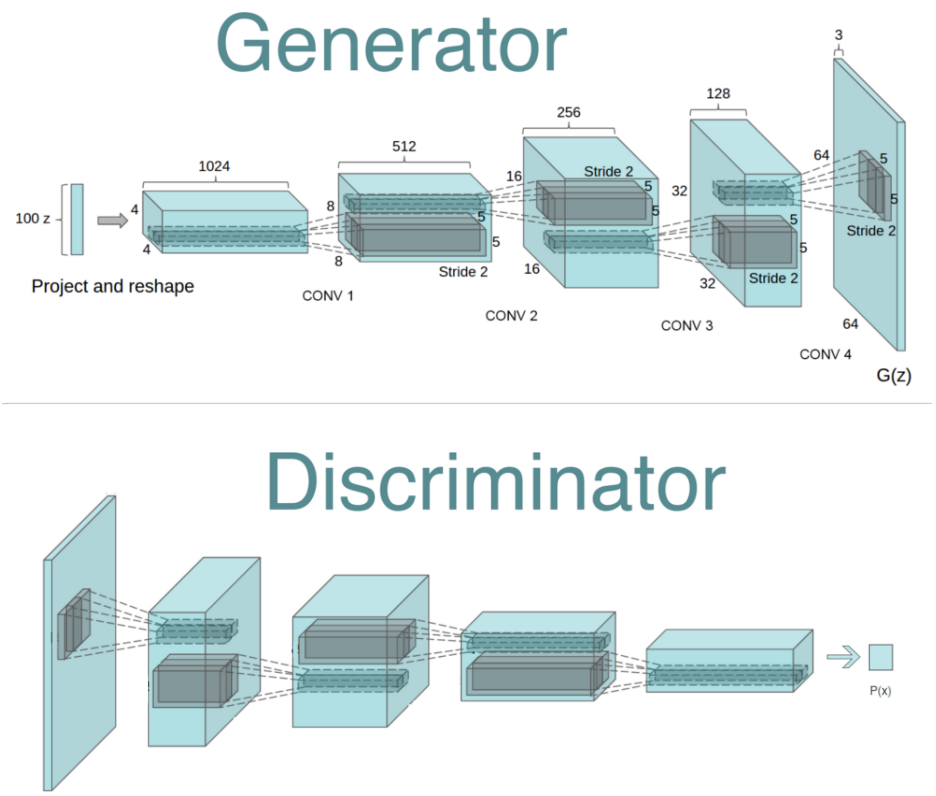


Figura 5.1: Redes que forman parte de una GAN.

Las GANs, se utilizan en el aprendizaje no supervisado, implementadas por un sistema de dos redes neuronales que compiten mutuamente en una especie de juego de suma cero. Esta técnica puede generar fotografías que parecen auténticas a observadores humanos. Por ejemplo, una fotografía sintética de un gato que consiga engañar al discriminador (una de las partes funcionales del algoritmo), es probable que lleve a una persona cualquiera a aceptarlo como una fotografía real.

5.2. Método

Una red genera los candidatos y otra los evalúa. Típicamente, la red generativa aprende a asignar elementos de un espacio latente a una distribución de datos determinada, mientras la red discriminativa diferencia entre elementos de la distribución de datos originales y los candidatos producidos por el generador. El objetivo del aprendizaje de la red generativa es aumentar el índice de error de la red discriminativa (o sea, “engañar” a la red discriminativa produciendo nuevos elementos sintéticos que parecen provenir de la distribución de datos auténticos, vease Figura 5.3 y Figura 5.2).

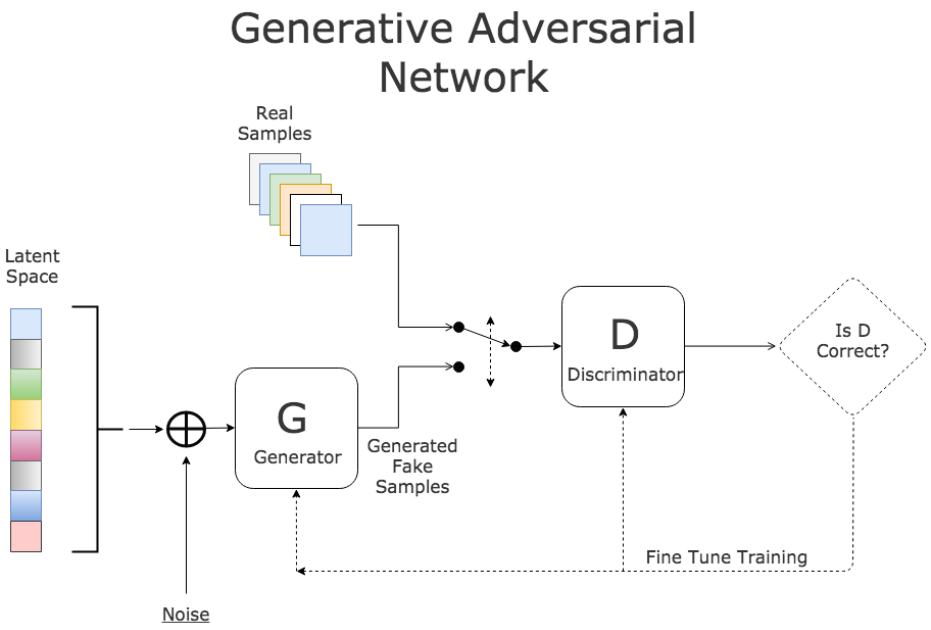


Figura 5.2: Funcionamiento de una GAN.

En la práctica, un conjunto de datos conocido sirve como el saber de partida para el discriminador. Entrenar al discriminador implica presentarle muestras del conjunto de datos, hasta que logra algún nivel de exactitud. Habitualmente, el generador está "sembrado" con una entrada aleatorizada que se escoge de un espacio latente predefinido (p. ej. una distribución normal multivariante). Después, las muestras sintetizadas por el generador son evaluadas por el discriminador. En ambas redes se aplica la retropropagación, de modo que el generador produce imágenes progresivamente mejores, mientras el discriminador se refina cada vez más a la hora de distinguir esas imágenes sintéticas. Los generadores son normalmente redes neuronales deconvolucionales, y los discriminadores son redes neuronales convolucionales. La idea de inferir modelos en un sistema competitivo (modelo versus discriminador) fue propuesta por Li, Gauci y Bruto en 2013. Su método se usa para inferencia conductista. Se denomina Aprendizaje de Turing, puesto que el esquema recuerda mucho al de un Test de Turing.

5.3. Aplicaciones

Las GANs se han utilizado para producir muestras de imágenes fotorrealistas de diseño industrial, de interiores, de ropa y complementos, o elementos para escenas de juegos de ordenador. Han aparecido informaciones de que Facebook ha llegado a utilizar este tipo de redes. Recientemente, algunas GAN han generado patrones de movimiento en vídeo. También se han utilizado para reconstruir modelos 3D a partir de imágenes 2D y para mejorar imágenes astronómicas.

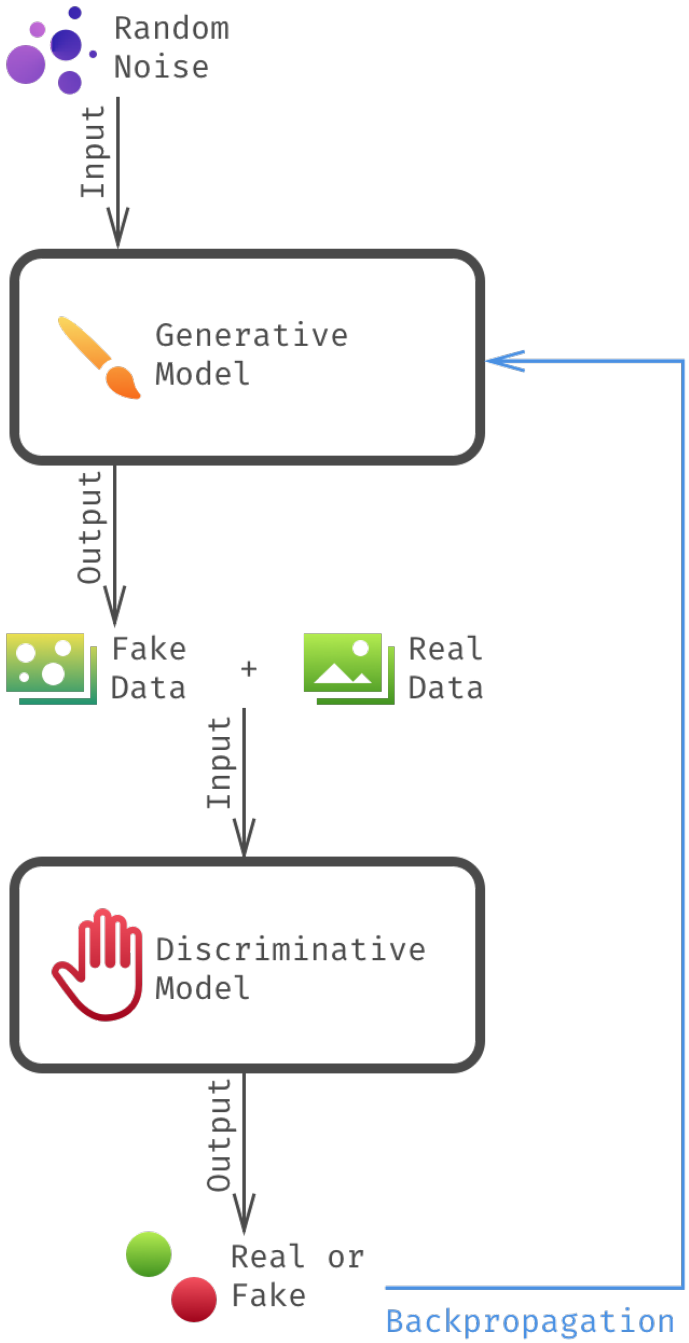


Figura 5.3: Funcionamiento de una GAN.

Las Siguientes imágenes muestran las diversas aplicaciones que toman ventajas de las GANs para su funcionamiento.

Entrada Salida Salida SRGAN Imagen Original

Figura 5.4: Aumento de la resolución de las imágenes [Ledig et al.,].

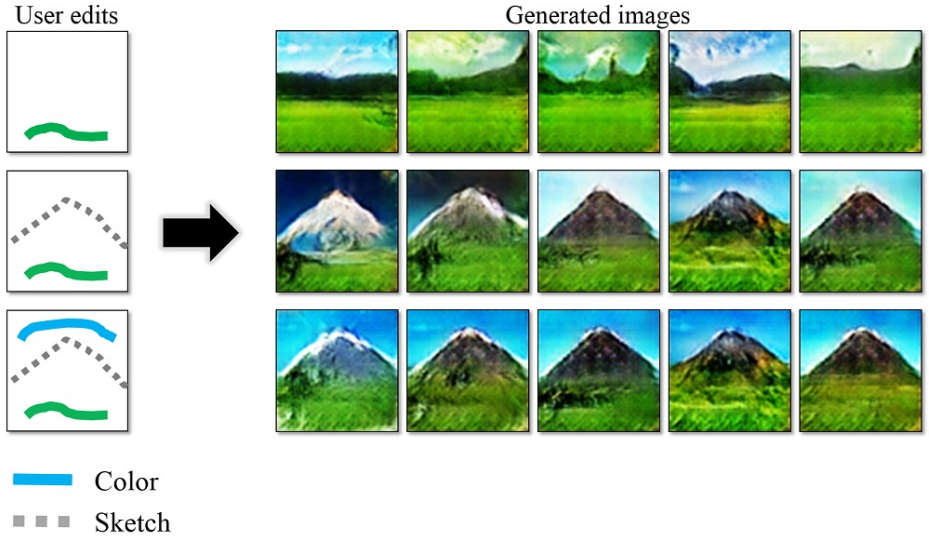


Figura 5.5: Generación Interactiva de imágenes [Zhu et al., 2016] .

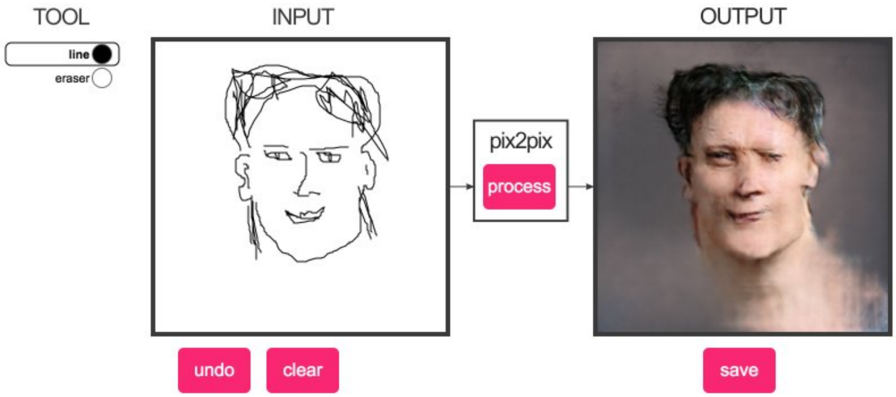


Figura 5.6: Traducción Imagen-Imagen [Isola et al., 2016].

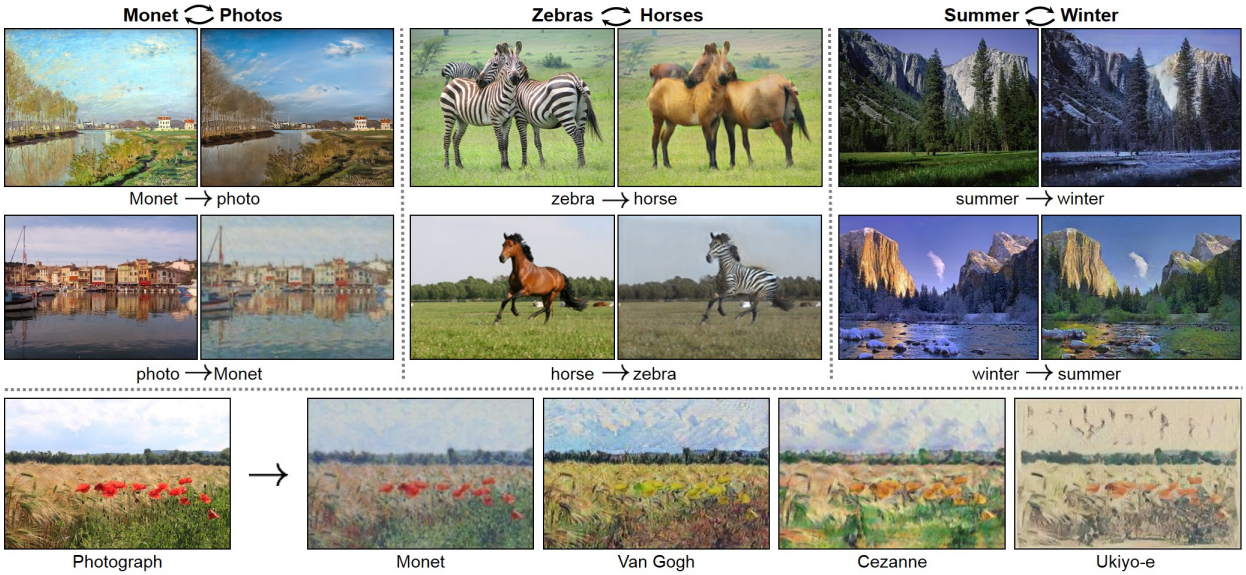


Figura 5.7: Cycle GAN [Zhu et al., 2017].

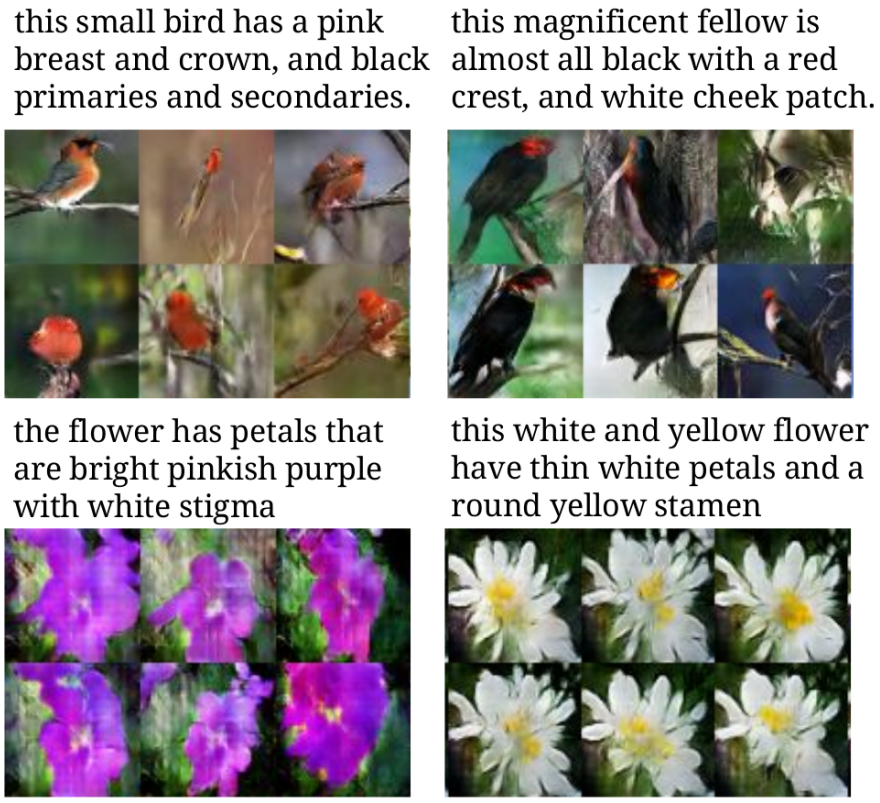


Figura 5.8: Traducción texto a imagen [Reed et al., 2016].

Capítulo 6

Herramientas de Terceros y Aplicaciones del DL

6.1. Herramientas de Clasificación en Linea

6.1.1. Clarifai

Figura 6.1: Clarifai

Clarifai nace como un grupo de investigación que participa en el desafió ImageNet 2013. Este desafió consiste en la elaboración de un modelo de clasificación capaz de identificar 1000 categorías diferentes. Para esto la organización responsable del desafió pone a disposición de los participantes y del publico general un dataset con mas de 1 Millón de imágenes divididas en 1000 categorías diferentes. Este dataset llamado ImageNet es considerado uno de los completos para trabajar con este tipo de problemas. Clarifai participa en este desafió obteniendo el primer lugar de ese año. Basaron su implementación en el uso de redes neuronales convolucionales. Hoy en día son una compañía enfocada a ofrecer soluciones basadas en ML y DL a diferentes empresas y/o personas. En sus inicios ofrecían el servicio de etiquetado de imágenes, se enviaba una imagen a su servidor y este devolvía una lista de etiquetas y probabilidades para cada etiqueta. En la actualidad han diversificado sus servicios y modelos de clasificación, de manera que ahora poseen modelos espe-

cializados en colores, NSFW, caras, logos texturas. Es un sistema de pago más orientado al uso por parte de empresas. Ofrece a su vez un a versión gratis que permite etiquetar un total de10 000 imágenes al mes. Su API se puede acceder desde gran variedad de lenguajes de programación. Las etiquetas que produzca para cada imagen están basadas en sus propias listas de miles de categorías.

6.1.2. Google Cloud Vision

La API Vision de Google Cloud permite que los desarrolladores comprendan el contenido de una imagen mediante el encapsulado de potentes modelos de aprendizaje automático en una API REST fácil de usar. La API clasifica imágenes rápidamente en miles de categorías (por ejemplo, “barco de vela”, “león” o “torre Eiffel”), detecta objetos y caras individuales dentro de las imágenes, además de buscar y leer palabras impresas en ellas. Esta plataforma tiene un flujo de trabajo similar a Clarifai, pero con diferentes modelos. Los modelos creados por Google, pueden estimar también detectar la ubicación de rostros en la imagen, detectar logotipos entre otros. Este servicio es gratuito hasta que se llegue a una determinada cantidad de consultas. Comparado con Clarifai, la mayor diferencia se encuentra en el numero de etiquetas que se producen con el algoritmo. Mientras Clarifai mantiene este numero constante, el Cloud Vision, devuelve un numero variable de etiquetas para cada imagen.

Bibliografía

[Chatfield et al., 2014] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. In *Proceedings of the British Machine Vision Conference*, Nottingham, UK. BMVA Press. 71

[Clarifai, 2015] Clarifai (2015). Clarifai: Amplifying intelligence. 58

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE. 68

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`. 64, 65, 66

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc. 81

[Géron, 2017] Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly. 5, 6, 7, 10, 11, 12, 13, 14, 15, 17, 19, 20, 21, 22, 23, 35, 37, 38, 39, 46, 47, 48, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 63, 65, 75, 76, 77, 78, 80

[Isola et al., 2016] Isola, P., Zhu, J., Zhou, T., and Efros, A. A. (2016). Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004. 86

[Jia et al., 2014] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*. 71

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. 58, 67, 69

[Ledig et al.,] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. Photo-realistic single image super-resolution using a generative adversarial network. 85

[Miguel Cazorla, 2003] Miguel Cazorla, Francisco Escolano, M. I. A. O. C. M. L. (2003). *Inteligencia Artificial Modelos Tecnicas y Areas de Aplicacion*. Thomson. 4

[Rangel, 2017] Rangel, J. C. (2017). Scene understanding for mobile robots exploiting deep learning techniques. 58, 61, 65, 66, 67, 68

[Reed et al., 2016] Reed, S. E., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., and Lee, H. (2016). Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396. 87

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. 68

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958. 66

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842. 67, 70

[Zhou et al., 2014] Zhou, B., Lapedriza, A., Xiao, J., Torralba, A., and Oliva, A. (2014). Learning deep features for scene recognition using places database. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc. 68, 69, 71

[Zhu et al., 2017] Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593. 86

[Zhu et al., 2016] Zhu, J.-Y., Krähenbühl, P., Shechtman, E., and Efros, A. A. (2016). Generative visual manipulation on the natural image manifold. In *Proceedings of European Conference on Computer Vision (ECCV)*. 85

Lista de Acrónimos

ANNs *Artificial Neural Networks*

API *Application Programming Interface*

Caffe *Convolutional Architecture for Fast Feature Embedding*

CNN *Convolutional Neural Networks*

DL *Deep Learning*

GAN *Generative Adversarial Networks*

GPU *Graphic Processing Unit*

ILSVRC *ImageNet Large Scale Visual Recognition Challenge*

kNN *k-Nearest Neighbors*

LSTM *Long Short-Term Memory*

LTU *linear threshold unit*

ML *Machine Learning*

RBF *radial basic function*

ReLUs *Rectifier Linear Units*

RFs *Random Forests*

RNN *Recurrent Neural Networks*

PCA *Principal Component Analysis*

SVM *Support Vectors Machine*