

TensorFlow



APRENDER EN 1 DIA

KRISHNA RUNGTA

TensorFlow en 1 día: Haz tu propia red neuronal

Por Krishna Rungta

Copyright 2018 - Todos los derechos reservados — Krishna Rungta

TODOS LOS DERECHOS RESERVADOS. Ninguna parte de esta publicación puede ser reproducida o transmitida en cualquier forma, electrónica o mecánica, incluyendo fotocopias, grabaciones, o por cualquier sistema de almacenamiento o recuperación de información sin el permiso expreso por escrito, fechado y firmado del autor.

Tabla de contenido

Capítulo 1: ¿Qué es el aprendizaje profundo?

1. [¿Qué es el aprendizaje profundo?](#)
2. [Proceso de aprendizaje profundo](#)
3. [Clasificación de Redes Neuronales](#)
4. [Tipos de Redes de Aprendizaje Profundo](#)
5. [Redes neuronales de alimentación](#)
6. [Redes neuronales recurrentes \(RNN\)](#)
7. [Redes neuronales convolucionales \(CNN\)](#)

Capítulo 2: Aprendizaje automático frente a aprendizaje profundo

1. [¿Qué es la IA?](#)
2. [¿Qué es ML?](#)
3. [¿Qué es el aprendizaje profundo?](#)
4. [Proceso de aprendizaje automático](#)
5. [Proceso de aprendizaje profundo](#)
6. [Automatizar la extracción de entidades mediante DL](#)
7. [Diferencia entre el aprendizaje automático y el aprendizaje profundo](#)
8. [¿Cuándo usar ML o DL?](#)

Capítulo 3: ¿Qué es TensorFlow?

1. [¿Qué es TensorFlow?](#)
2. [Historia de TensorFlow](#)
3. [Arquitectura de TensorFlow](#)
4. [¿Dónde puede correr Tensorflow?](#)

5. [Introducción a los componentes de TensorFlow](#)
6. [¿Por qué TensorFlow es popular?](#)
7. [Lista de algoritmos prominentes soportados por TensorFlow](#)

Capítulo 4: Comparación de bibliotecas de aprendizaje profundo

1. [8 Mejores Bibliotecas/Marco de aprendizaje profundo](#)
2. [MICROSOFT COGNITIVE TOOLKIT \(CNTK\)](#)
3. [TensorFlow vs Theanos vs Torch vs Keras vs infer.net vs CNTK vs MXNet vs Caffe: Diferencias clave](#)

Capítulo 5: Cómo descargar e instalar TensorFlow Windows y Mac

1. [Versiones de TensorFlow](#)
2. [Instalar Anaconda](#)
3. [Crear un archivo.yml para instalar Tensorflow y dependencias](#)
4. [Lanzamiento de Jupyter Notebook](#)
5. [Jupyter con el entorno principal de conda](#)

Capítulo 6: Tutorial de Cuaderno de Jupyter

1. [¿Qué es Jupyter Notebook?](#)
2. [Aplicación para portátiles Jupyter](#)
3. [Cómo usar Jupyter](#)

Capítulo 7: Tensorflow en AWS

1. [PARTE 1: Configurar un par de claves](#)
2. [PARTE 2: Configurar un grupo de seguridad](#)
3. [Inicie su instancia \(usuarios de Windows\)](#)

4. [Parte 4: Instalar Docker](#)
5. [Parte 5: Instalar Jupyter](#)
6. [Parte 6: Cerrar conexión](#)

Capítulo 8: Conceptos básicos de TensorFlow: Tensor, Forma, Tipo, Gráfico, Sesiones y Operadores

1. [¿Qué es un Tensor?](#)
2. [Representación de un tensor](#)
3. [Tipos de Tensor](#)
4. [Forma del tensor](#)
5. [Tipo de datos](#)
6. [Creando operador](#)
7. [Variables](#)

Capítulo 9: Tensorboard: Visualización de gráficos con ejemplo

Capítulo 10: NumPy

1. [¿Qué es NumPy?](#)
2. [¿Por qué usar NumPy?](#)
3. [¿Cómo instalar NumPy?](#)
4. [Operaciones matemáticas en una matriz](#)
5. [Forma de matriz](#)
6. [np.zeros y np.ones](#)
7. [Reformar y aplanar datos](#)
8. [hstack y vstack](#)

Capítulo 11: Pandas

1. [¿Qué es Pandas?](#)
2. [¿Por qué usar Pandas?](#)

3. [¿Cómo instalar Pandas?](#)
4. [¿Qué es un marco de datos?](#)
5. [¿Qué es una serie?](#)
6. [Concatenación](#)

Capítulo 12: Scikit-Learn

1. [¿Qué es SciKit-learn?](#)
2. [Descargar e instalar scikit-learn](#)
3. [Aprendizaje automático con scikit-learn](#)
4. [Paso 1\) Importar los datos](#)
5. [Paso 2\) Crear el conjunto de tren/prueba](#)
6. [Paso 3\) Construir la tubería](#)
7. [Paso 4\) Uso de nuestra tubería en una búsqueda de cuadrícula](#)

Capítulo 13: Regresión lineal

1. [Regresión lineal](#)
2. [Cómo entrenar un modelo de regresión lineal](#)
3. [Cómo entrenar una Regresión Lineal con TensorFlow](#)
4. [Pandas](#)
5. [Solución Numpy](#)
6. [Solución Tensorflow](#)

Capítulo 14: Estudio de caso de regresión lineal

1. [Resumen de estadísticas](#)
2. [Descripción general de las facetas](#)
3. [Inmersión profunda de facetas](#)
4. [Instalar faceta](#)
5. [Resumen](#)
6. [Gráfico](#)

7. [Inmersión profunda de facetas](#)

Capítulo 15: Clasificador lineal en TensorFlow

1. [¿Qué es el clasificador lineal?](#)
2. [¿Cómo funciona el clasificador binario?](#)
3. [¿Cómo medir el rendimiento de Linear Classifier?](#)
4. [Clasificador lineal con TensorFlow](#)

Capítulo 16: Métodos del núcleo

1. [¿Por qué necesita Kernel Methods?](#)
2. [¿Qué es un kernel en el aprendizaje automático?](#)
3. [Tipo de métodos de kernel](#)
4. [Clasificador de Kernel Gaussian con TensorFlow](#)

Capítulo 17: ANN de TensorFlow (Red Neural Artificial)

1. [¿Qué es la red neuronal artificial?](#)
2. [Arquitectura de red neuronal](#)
3. [Limitaciones de la red neuronal](#)
4. [Ejemplo de red neuronal en TensorFlow](#)
5. [Entrenar una red neuronal con TensorFlow](#)

Capítulo 18: ConvNet (red neuronal convolucional): Clasificación de imágenes de TensorFlow

1. [¿Qué es la red neuronal convolucional?](#)
2. [Arquitectura de una red neuronal convolucional](#)
3. [Componentes de Convnets](#)
4. [Tren CNN con TensorFlow](#)

Capítulo 19: Autoencoder con TensorFlow

1. [¿Qué es un autocodificador?](#)
2. [¿Cómo funciona Autoencoder?](#)
3. [Ejemplo de Autocodificador apilado](#)
4. [Crear un autocodificador con TensorFlow](#)

Capítulo 20: RNN (red neuronal recurrente) TensorFlow

1. [¿Qué necesitamos un RNN?](#)
2. [¿Qué es RNN?](#)
3. [Construye un RNN para predecir series temporales en TensorFlow](#)

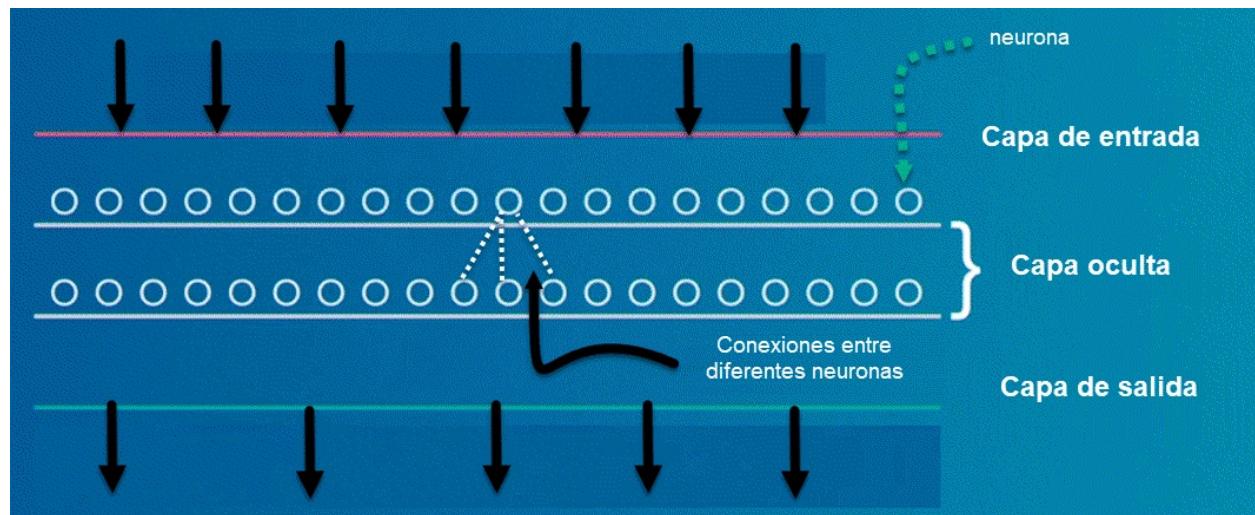
Capítulo 1: ¿Qué es el aprendizaje profundo?

¿Qué es el aprendizaje profundo?

El aprendizaje profundo es un software informático que **imita la red de neuronas en el cerebro**. Es un subconjunto de aprendizaje automático y se llama aprendizaje profundo porque hace uso de **redes neuronales**.

Los algoritmos de aprendizaje profundo se construyen con capas conectadas.

- La primera capa se llama capa de entrada
- La última capa se llama Capa de salida
- Todas las capas intermedias se denominan capas ocultas. La palabra profunda significa que la red se une a las neuronas en más de dos capas.



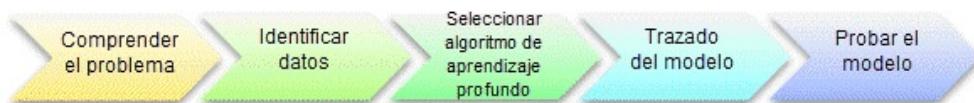
Cada capa oculta se compone de neuronas. Las neuronas están

conectadas entre sí. La neurona procesará y luego propagará la señal de entrada que recibe la capa por encima de ella. La fuerza de la señal dada a la neurona en la siguiente capa depende del peso, el sesgo y la función de activación.

La red consume grandes cantidades de datos de entrada y los opera a través de múltiples capas; la red puede aprender características cada vez más complejas de los datos en cada capa.

Proceso de aprendizaje profundo

Una red neuronal profunda proporciona precisión de vanguardia en muchas tareas, desde la detección de objetos hasta el reconocimiento de voz. Pueden aprender automáticamente, sin conocimientos predefinidos codificados explícitamente por los programadores.



Para comprender la idea del aprendizaje profundo, imagina una familia, con un bebé y padres. El niño señala objetos con su dedo meñique y siempre dice la palabra “gato”. Como sus padres están preocupados por su educación, le siguen diciendo “Sí, eso es un gato” o “No, eso no es un gato”. El bebé persiste en señalar objetos, pero se vuelve más preciso con 'gatos'. El niño pequeño, en el fondo, no sabe por qué puede decir que es un gato o no. Él acaba de aprender jerarquías características complejas que vienen con un gato mirando a la mascota en general y seguir centrándose en detalles como las colas o la nariz antes de tomar su decisión.

Una red neuronal funciona igual. Cada capa representa un nivel más profundo de conocimiento, es decir, la jerarquía del conocimiento. Una red neuronal con cuatro capas aprenderá una característica más compleja que con la de dos capas.

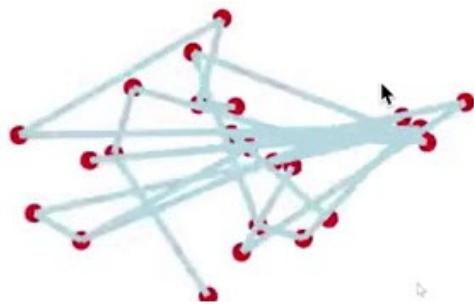
El aprendizaje ocurre en dos fases.

- La primera fase consiste en aplicar una transformación no lineal de la entrada y crear un modelo estadístico como salida.
- La segunda fase tiene como objetivo mejorar el modelo con un

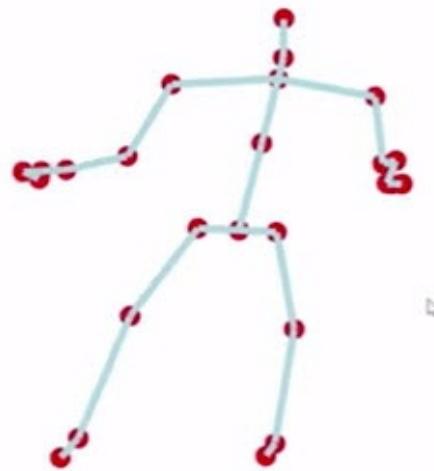
método matemático conocido como derivado.

La red neuronal repite estas dos fases cientos a miles de tiempo hasta que ha alcanzado un nivel tolerable de precisión. La repetición de esta bifase se llama iteración.

Para dar un ejemplo, eche un vistazo al movimiento de abajo, el modelo está tratando de aprender a bailar. Después de 10 minutos de entrenamiento, el modelo no sabe cómo bailar, y parece un garabato.



Después de 48 horas de aprendizaje, la computadora domina el arte del baile.



Clasificación de Redes Neuronales

Red neuronal superficial: La red neural superficial tiene sólo una capa oculta entre la entrada y la salida.

Red neuronal profunda: Las redes neuronales profundas tienen más de una capa. Por ejemplo, el modelo de Google LeNet para el reconocimiento de imágenes cuenta 22 capas.

Hoy en día, el aprendizaje profundo se utiliza de muchas maneras como un coche sin conductor, teléfono móvil, motor de búsqueda de Google, detección de fraude, televisión, etc.

Tipos de Redes de Aprendizaje Profundo

Un gráfico mayormente completo de

Redes Neuronales

©2016 Fjodor van Veen - asimovinstitute.org

Backfed Input Cell

Input Cell

Noisy Input Cell

Hidden Cell

Probabilistic Hidden Cell

Spiking Hidden Cell

Output Cell

Match Input Output Cell

Recurrent Cell

Memory Cell

Different Memory Cell

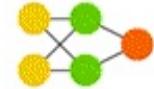
Kernel

Convolution or Pool

Perceptron (P)



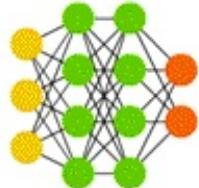
Feed Forward (FF)



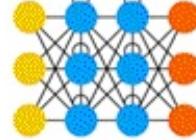
Radial Basis Network (RBF)



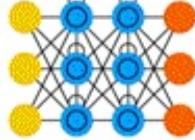
Deep Feed Forward (DFF)



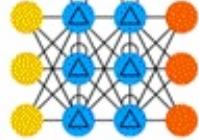
Recurrent Neural Network (RNN)



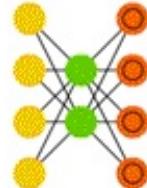
Long / Short Term Memory (LSTM)



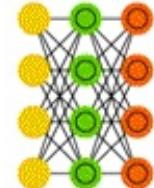
Gated Recurrent Unit (GRU)



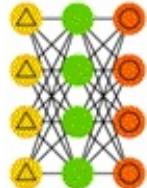
Auto Encoder (AE)



Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)

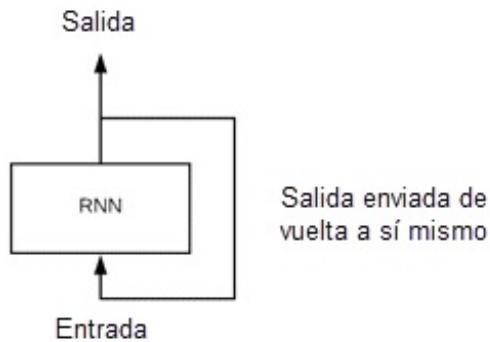


Redes neuronales de alimentación

El tipo más simple de red neuronal artificial. Con este tipo de arquitectura, la información fluye en una sola dirección, hacia adelante. Significa que los flujos de la información comienzan en la capa de entrada, van a las capas “ocultas”, y terminan en la capa de salida. La red no tiene un bucle. La información se detiene en las capas de salida.

Redes neuronales recurrentes (RNN)

RNN es una red neuronal de varias capas que puede almacenar información en nodos de contexto, lo que le permite aprender secuencias de datos y generar un número u otra secuencia. En palabras simples es un artificial redes neuronales cuyas conexiones entre neuronas incluyen bucles. Los RNN son muy adecuados para procesar secuencias de entradas.



Ejemplo, si la tarea es predecir la siguiente palabra en la frase “¿Quieres un.....?

- Las neuronas RNN recibirán una señal que apunta al inicio de la oración.
- La red recibe la palabra “Do” como entrada y produce un vector del número. Este vector se alimenta de nuevo a la neurona para proporcionar una memoria a la red. Esta etapa ayuda a la red a recordar que recibió “Do” y lo recibió en la primera posición.
- La red procederá de manera similar a las siguientes palabras. Se necesita la palabra “tú” y “quieres”. El estado de las neuronas se actualiza al recibir cada palabra.

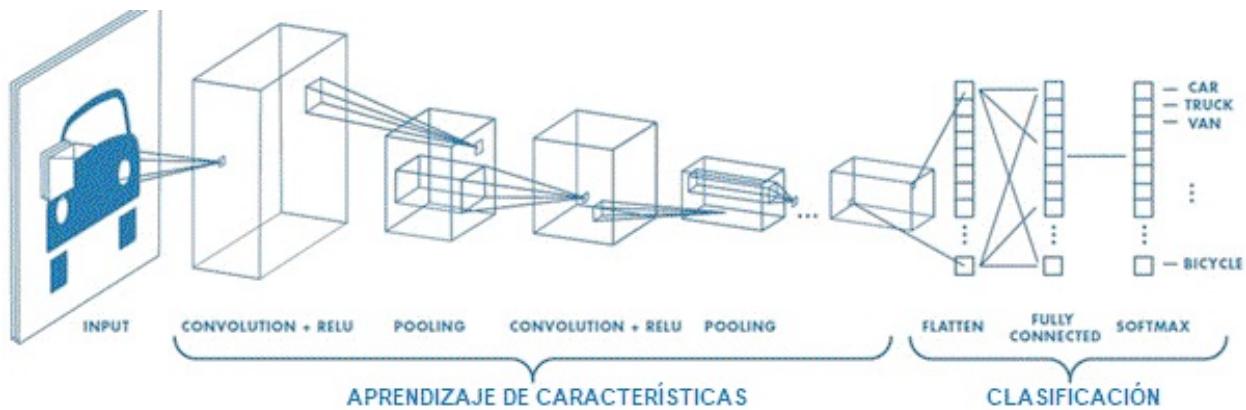
- La etapa final ocurre después de recibir la palabra “a”. La red neuronal proporcionará una probabilidad para cada palabra en inglés que se puede usar para completar la oración. Un RNN bien entrenado probablemente asigna una alta probabilidad a “café”, “bebida”, “hamburguesa”, etc.

Usos comunes de RNN

- Ayudar a los comerciantes de valores a generar informes analíticos
- Detectar anomalías en el contrato de estados financieros
- Detectar transacciones fraudulentas con tarjeta de crédito
- Proporcionar un título para las imágenes
- Chatbots de potencia
- Los usos estándar de RNN ocurren cuando los profesionales están trabajando con datos de series temporales o secuencias (por ejemplo, grabaciones de audio o texto).

Redes neuronales convolucionales (CNN)

CNN es una red neuronal de varias capas con una arquitectura única diseñada para extraer características cada vez más complejas de los datos en cada capa para determinar la salida. Los CNN son muy adecuados para tareas perceptivas.



CNN se usa principalmente cuando hay un conjunto de datos no estructurados (por ejemplo, imágenes) y los profesionales necesitan extraer información de él

Por ejemplo, si la tarea es predecir un título de imagen:

- La CNN recibe una imagen de digamos un gato, esta imagen, en términos informáticos, es una colección del píxel. Generalmente, una capa para la imagen en escala de grises y tres capas para una imagen en color.
- Durante el aprendizaje de características (es decir, capas ocultas), la red identificará características únicas, por ejemplo, la cola del gato, la oreja, etc.
- Cuando la red aprendió a reconocer una imagen, puede proporcionar

una probabilidad para cada imagen que conoce. La etiqueta con la mayor probabilidad se convertirá en la predicción de la red.

Aprendizaje de refuerzo

El aprendizaje de refuerzo es un subcampo del aprendizaje automático en el que los sistemas son entrenados recibiendo “recompensas” virtuales o “castigos”, esencialmente aprendiendo por ensayo y error. DeepMind de Google ha utilizado el aprendizaje de refuerzo para vencer a un campeón humano en los juegos Go. El aprendizaje de refuerzo también se utiliza en los videojuegos para mejorar la experiencia de juego proporcionando bot más inteligente.

Uno de los algoritmos más famosos son:

- Q-Learning
- Red Q profunda
- Estado-Acción-Premio-Acción Estado-Estado (SARSA)
- Degradado de política determinista profundo (DDPG)

Aplicaciones / Ejemplos de aplicaciones de aprendizaje profundo

Inteligencia Artificial en Finanzas: El sector de la tecnología financiera ya ha comenzado a utilizar la IA para ahorrar tiempo, reducir costos y agregar valor. El aprendizaje profundo está cambiando la industria crediticia mediante el uso de calificaciones crediticias más sólidas. Los responsables de la toma de decisiones crediticias pueden utilizar IA para aplicaciones sólidas de crédito para lograr una evaluación de riesgos más rápida y precisa, utilizando inteligencia artificial para tener en cuenta el carácter y la capacidad de los solicitantes.

Underwrite es una empresa Fintech que proporciona una solución de inteligencia artificial para la empresa de creadores de crédito. underwrite.ai utiliza AI para detectar qué solicitante es más probable que pague un préstamo. Su enfoque supera radicalmente los métodos tradicionales.

IA en HR: Under Armour, una empresa de ropa deportiva revoluciona la contratación y moderniza la experiencia del candidato con la ayuda de AI. De hecho, Under Armour Reduce el tiempo de contratación para sus tiendas minoristas en un 35%. Under Armour se enfrentó a un creciente interés de popularidad en 2012. Tenían, en promedio, 30000 currículos al mes. Leer todas esas solicitudes y comenzar a iniciar el proceso de selección y entrevista estaba tomando demasiado tiempo. El largo proceso de conseguir que las personas contratadas y adjuntas impactó la capacidad de Under Armour de tener sus tiendas de venta al por menor con personal completo, rampas y listas para operar.

En ese momento, Under Armour tenía toda la tecnología de recursos humanos 'must have' en su lugar, como soluciones transaccionales para el

abastecimiento, la aplicación, el seguimiento y la incorporación, pero esas herramientas no eran lo suficientemente útiles. Bajo blindaje elegir **HireVue**, un proveedor de inteligencia artificial para soluciones de recursos humanos, tanto para entrevistas bajo demanda como en directo. Los resultados fueron un farol; lograron disminuir un 35% el tiempo para llenar. A cambio, contrataron personal de mayor calidad.

Inteligencia Artificial en Marketing: La IA es una herramienta valiosa para la gestión del servicio al cliente y los desafíos de personalización. El reconocimiento de voz mejorado en la gestión del centro de llamadas y el enrutamiento de llamadas como resultado de la aplicación de técnicas de IA permite una experiencia más fluida para los clientes.

Por ejemplo, el análisis de aprendizaje profundo del audio permite a los sistemas evaluar el tono emocional del cliente. Si el cliente está respondiendo mal al chatbot de inteligencia artificial, el sistema puede ser redireccionado la conversación a operadores humanos reales que se hacen cargo del problema.

A parte de los tres ejemplos anteriores, la IA se utiliza ampliamente en otros sectores e industrias.

¿Por qué es importante el aprendizaje profundo?

El aprendizaje profundo es una poderosa herramienta para hacer de la predicción un resultado procesable. El aprendizaje profundo sobresale en el descubrimiento de patrones (aprendizaje no supervisado) y la predicción basada en el conocimiento. El big data es el combustible para el aprendizaje profundo. Cuando ambos se combinan, una organización puede obtener resultados sin precedentes en términos de productividad, ventas, administración e innovación.

El aprendizaje profundo puede superar el método tradicional. Por ejemplo, los algoritmos de aprendizaje profundo son 41% más precisos que los algoritmos de aprendizaje automático en la clasificación de imágenes, 27% más precisos en reconocimiento facial y 25% en reconocimiento de voz.

Limitaciones del aprendizaje profundo

Etiquetado de datos

La mayoría de los modelos de IA actuales se capacitan a través del “aprendizaje supervisado”. Significa que los humanos deben etiquetar y categorizar los datos subyacentes, que pueden ser una tarea considerable y propensa a errores. Por ejemplo, las empresas que desarrollan tecnologías de autoconducción contratan a cientos de personas para anotar manualmente las horas de vídeo de los vehículos prototipos para ayudar a entrenar estos sistemas.

Obtener enormes conjuntos de datos de capacitación

Se ha demostrado que técnicas simples de aprendizaje profundo como la CNN pueden, en algunos casos, imitar el conocimiento de expertos en medicina y otros campos. Sin embargo, la actual ola de aprendizaje automático requiere conjuntos de datos de capacitación que no solo estén etiquetados sino que también sean suficientemente amplios y universales.

Los métodos de aprendizaje profundo requerían miles de observaciones para que los modelos fueran relativamente buenos en las tareas de clasificación y, en algunos casos, millones para que realizaran a nivel humano. Sin sorpresa, el aprendizaje profundo es famoso en las empresas tecnológicas gigantes; están usando big data para acumular petabytes de datos. Les permite crear un modelo de aprendizaje profundo impresionante y altamente preciso.

Explicar un problema

Los modelos grandes y complejos pueden ser difíciles de explicar, en

términos humanos. Por ejemplo, por qué se obtuvo una decisión en particular. Es una de las razones por las que la aceptación de algunas herramientas de inteligencia artificial es lenta en áreas de aplicación donde la interpretabilidad es útil o de hecho necesaria.

Además, a medida que se amplíe la aplicación de la IA, los requisitos reglamentarios también podrían impulsar la necesidad de modelos de IA más explicables.

Resumen

El aprendizaje profundo es el nuevo estado de la inteligencia artificial. La arquitectura de aprendizaje profundo se compone de una capa de entrada, capas ocultas y una capa de salida. La palabra deep significa que hay más de dos capas completamente conectadas.

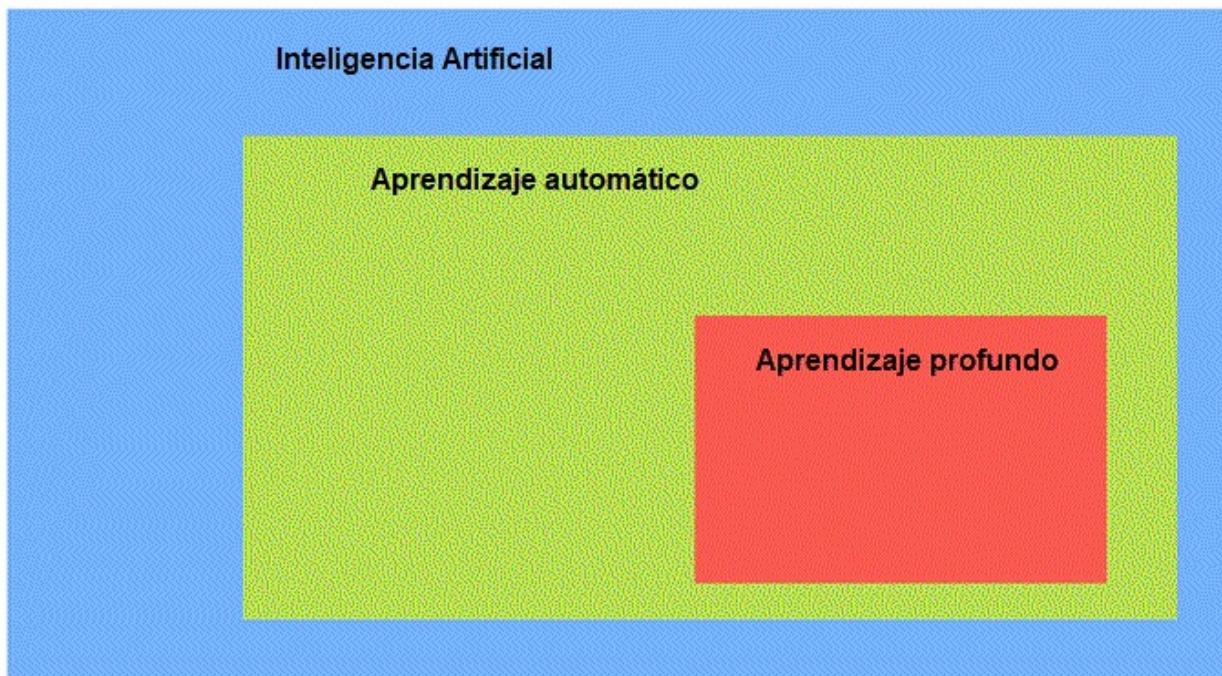
Hay una gran cantidad de red neuronal, donde cada arquitectura está diseñada para realizar una tarea dada. Por ejemplo, CNN funciona muy bien con imágenes, RNN proporciona resultados impresionantes con series temporales y análisis de texto.

El aprendizaje profundo ahora está activo en diferentes campos, desde las finanzas hasta el marketing, la cadena de suministro y el marketing. Las grandes empresas son las primeras en utilizar el aprendizaje profundo porque ya tienen un gran conjunto de datos. El aprendizaje profundo requiere contar con un extenso conjunto de datos de capacitación.

Capítulo 2: Aprendizaje automático frente a aprendizaje profundo

¿Qué es la IA?

La inteligencia artificial está impartiendo una capacidad cognitiva a una máquina. El punto de referencia para la IA es la inteligencia humana con respecto al razonamiento, el habla y la visión. Este punto de referencia está muy lejos en el futuro.



La IA tiene tres niveles diferentes:

1. **AI estrecho:** Se dice que una inteligencia artificial es estrecha cuando la máquina puede realizar una tarea específica mejor que un

humano. La investigación actual de IA está aquí ahora

2. **Inteligencia Artificial general:** Una inteligencia artificial alcanza el estado general cuando puede realizar cualquier tarea intelectual con el mismo nivel de precisión que lo haría un ser humano
3. **IA activa:** Una IA está activa cuando puede vencer a los humanos en muchas tareas

Los primeros sistemas de IA utilizaron la coincidencia de patrones y sistemas expertos.

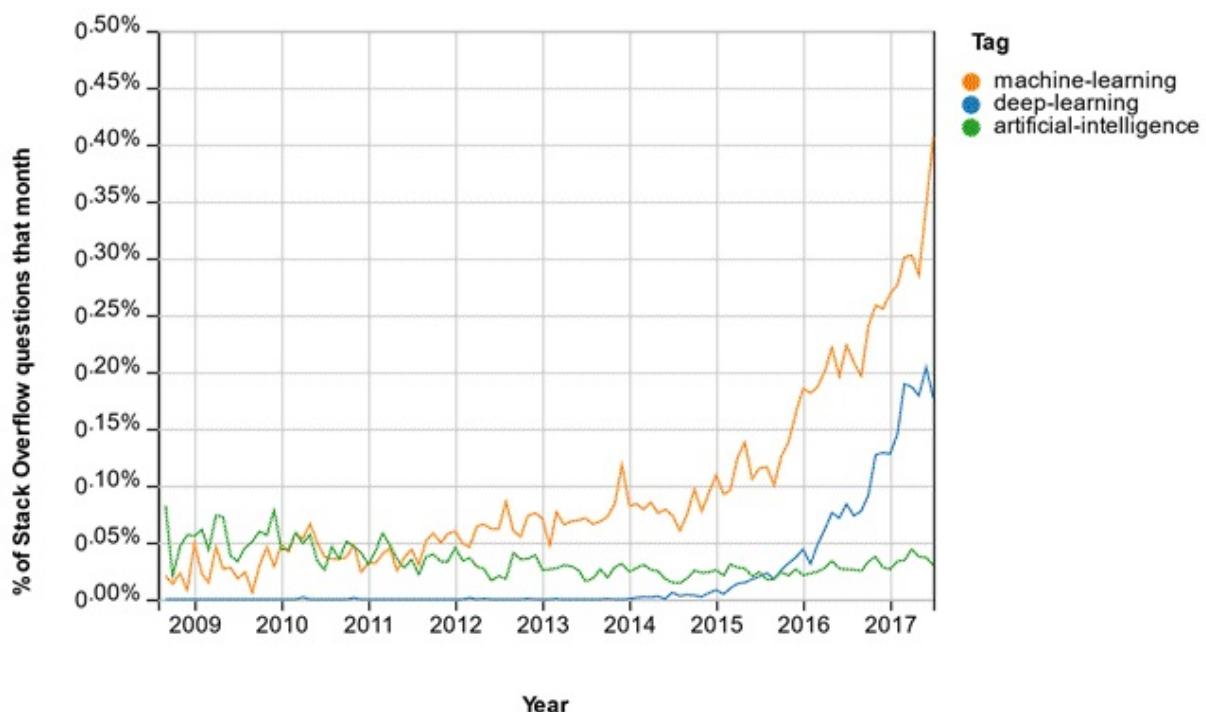
¿Qué es ML?

El aprendizaje automático es la mejor herramienta hasta ahora para analizar, comprender e identificar un patrón en los datos. Una de las ideas principales detrás del aprendizaje automático es que el ordenador puede ser entrenado para automatizar tareas que serían exhaustivas o imposibles para un ser humano. La clara brecha del análisis tradicional es que el aprendizaje automático puede tomar decisiones con una intervención humana mínima.

El aprendizaje automático utiliza datos para alimentar un algoritmo que puede entender la relación entre la entrada y la salida. Cuando la máquina terminó de aprender, puede predecir el valor o la clase de nuevo punto de datos.

¿Qué es el aprendizaje profundo?

El aprendizaje profundo es un software informático que imita la red de neuronas en un cerebro. Es un subconjunto de aprendizaje automático y se llama aprendizaje profundo porque hace uso de redes neuronales profundas. La máquina utiliza diferentes capas para aprender de los datos. La profundidad del modelo está representada por el número de capas del modelo. El aprendizaje profundo es el nuevo estado del arte en términos de IA. En el aprendizaje profundo, la fase de aprendizaje se realiza a través de una red neuronal. Una red neuronal es una arquitectura donde las capas se apilan una encima de la otra



Proceso de aprendizaje automático

Imagine que está destinado a construir un programa que reconozca objetos. Para entrenar el modelo, utilizará un **clasificador**. Un clasificador utiliza las entidades de un objeto para intentar identificar la clase a la que pertenece.

En el ejemplo, el clasificador será entrenado para detectar si la imagen es una:

- Bicicleta
- Barco
- Coche
- Avión

Los cuatro objetos anteriores son la clase que el clasificador tiene que reconocer. Para construir un clasificador, debe tener algunos datos como entrada y asignarle una etiqueta. El algoritmo tomará estos datos, encontrará un patrón y luego lo clasificará en la clase correspondiente.

Esta tarea se denomina **aprendizaje supervisado**. En el aprendizaje supervisado, los datos de formación que se alimentan al algoritmo incluyen una etiqueta.

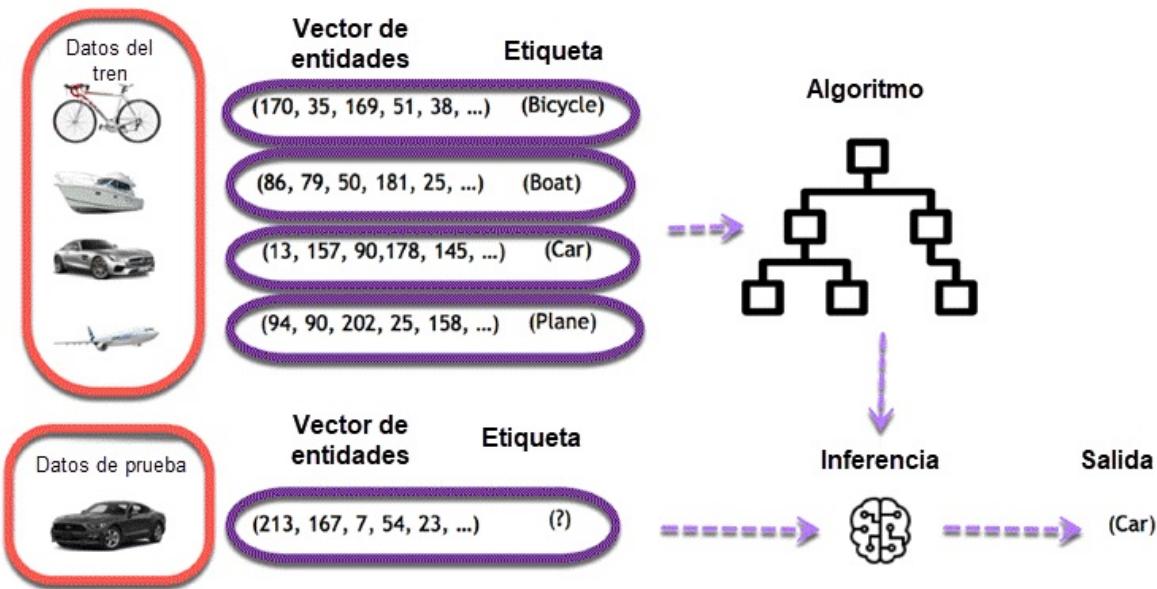
El entrenamiento de un algoritmo requiere seguir algunos pasos estándar:

- Recopilar los datos
- Entrenar al clasificador
- Hacer predicciones

El primer paso es necesario, elegir los datos correctos hará que el

algoritmo sea exitoso o un fracaso. Los datos que elija para entrenar el modelo se denominan función. En el ejemplo de objeto, las entidades son los píxeles de las imágenes.

Cada imagen es una fila en los datos, mientras que cada píxel es una columna. Si la imagen tiene un tamaño de 28x28, el conjunto de datos contiene 784 columnas (28x28). En la imagen siguiente, cada imagen se ha transformado en un vector de entidades. La etiqueta indica al equipo qué objeto está en la imagen.



El objetivo es utilizar estos datos de entrenamiento para clasificar el tipo de objeto. El primer paso consiste en crear las columnas de entidades. Luego, el segundo paso implica elegir un algoritmo para entrenar el modelo. Cuando se realiza el entrenamiento, el modelo predice qué imagen corresponde a qué objeto.

Después de eso, es fácil usar el modelo para predecir nuevas imágenes. Para cada nueva imagen se alimenta en el modelo, la máquina predice la clase a la que pertenece. Por ejemplo, una imagen completamente nueva sin etiqueta está pasando por el modelo. Para un ser humano, es trivial

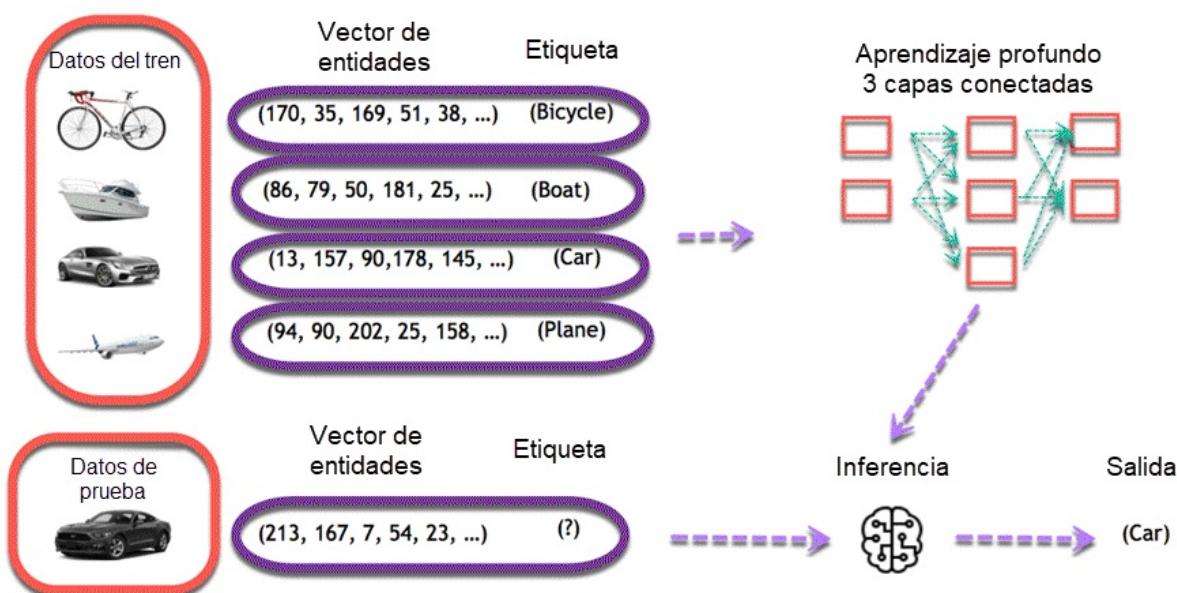
visualizar la imagen como un automóvil. La máquina utiliza sus conocimientos previos para predecir también la imagen es un coche.

Proceso de aprendizaje profundo

En el aprendizaje profundo, la fase de aprendizaje se realiza a través de una red neuronal. Una red neuronal es una arquitectura donde las capas se apilan una encima de la otra.

Considere el mismo ejemplo de imagen de arriba. El conjunto de entrenamiento se alimentaría a una red neuronal

Cada entrada entra en una neurona y se multiplica por un peso. El resultado de la multiplicación fluye a la siguiente capa y se convierte en la entrada. Este proceso se repite para cada capa de la red. La capa final se denomina capa de salida; proporciona un valor real para la tarea de regresión y una probabilidad de cada clase para la tarea de clasificación. La red neuronal utiliza un algoritmo matemático para actualizar los pesos de todas las neuronas. La red neuronal está completamente entrenada cuando el valor de los pesos da una salida cercana a la realidad. Por ejemplo, una red neuronal bien entrenada puede reconocer el objeto en una imagen con mayor precisión que la red neuronal tradicional.



Automatizar la extracción de entidades mediante DL

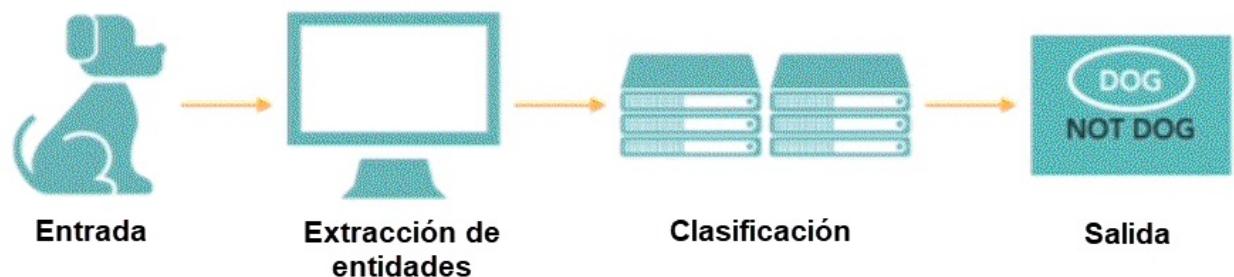
Un dataset puede contener entre una docena y cientos de entidades. El sistema aprenderá de la relevancia de estas características. Sin embargo, no todas las entidades son significativas para el algoritmo. Una parte crucial del aprendizaje automático es encontrar un conjunto relevante de características para hacer que el sistema aprenda algo.

Una forma de realizar esta parte en el aprendizaje automático es utilizar la extracción de entidades. La extracción de entidades combina entidades existentes para crear un conjunto de entidades más relevante. Se puede hacer con PCA, T-SNE o cualquier otro algoritmo de reducción de dimensionalidad.

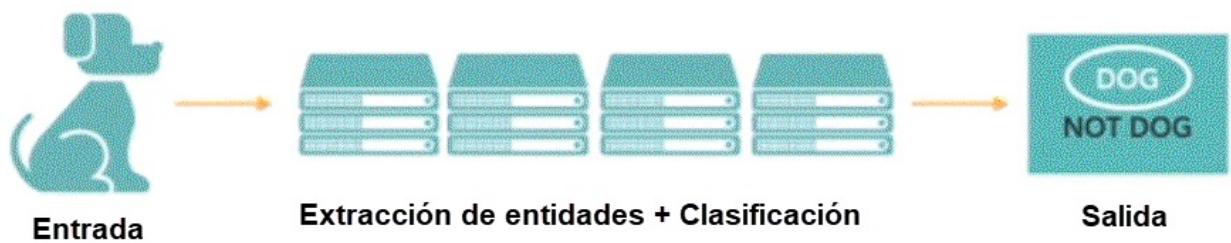
Por ejemplo, un procesamiento de imágenes, el practicante necesita extraer la característica manualmente en la imagen como los ojos, la nariz, los labios, etc. Esas entidades extraídas se alimentan al modelo de clasificación.

El aprendizaje profundo resuelve este problema, especialmente para una red neuronal convolucional. La primera capa de una red neuronal aprenderá pequeños detalles de la imagen; las siguientes capas combinarán los conocimientos previos para hacer información más compleja. En la red neuronal convolucional, la extracción de características se realiza con el uso del filtro. La red aplica un filtro a la imagen para ver si hay una coincidencia, es decir, la forma de la entidad es idéntica a una parte de la imagen. Si hay una coincidencia, la red utilizará este filtro. Por lo tanto, el proceso de extracción de funciones se realiza automáticamente.

APRENDIZAJE AUTOMÁTICO TRANDICIONAL



APRENDIZAJE PROFUNDO



Diferencia entre el aprendizaje automático y el aprendizaje profundo

	Aprendizaje automático	Aprendizaje profundo
Dependencias de datos	Excelentes prestaciones en un conjunto de datos pequeño/mediano	Excelente rendimiento en un gran conjunto de datos
Dependencias de hardware	Trabajar en una máquina de gama baja.	Requiere máquina potente, preferiblemente con GPU: DL realiza una cantidad significativa de multiplicación de matriz
Ingeniería de características	Necesidad de comprender las características que representan los datos	No es necesario entender la mejor característica que representa los datos
Tiempo de ejecución	De pocos minutos a horas	Hasta semanas. La red neuronal necesita calcular un número significativo de pesos
Interpretabilidad	Algunos algoritmos son fáciles de interpretar (logística, árbol de decisiones), algunos son casi imposibles (SVM, XgBoost)	Difícil a imposible

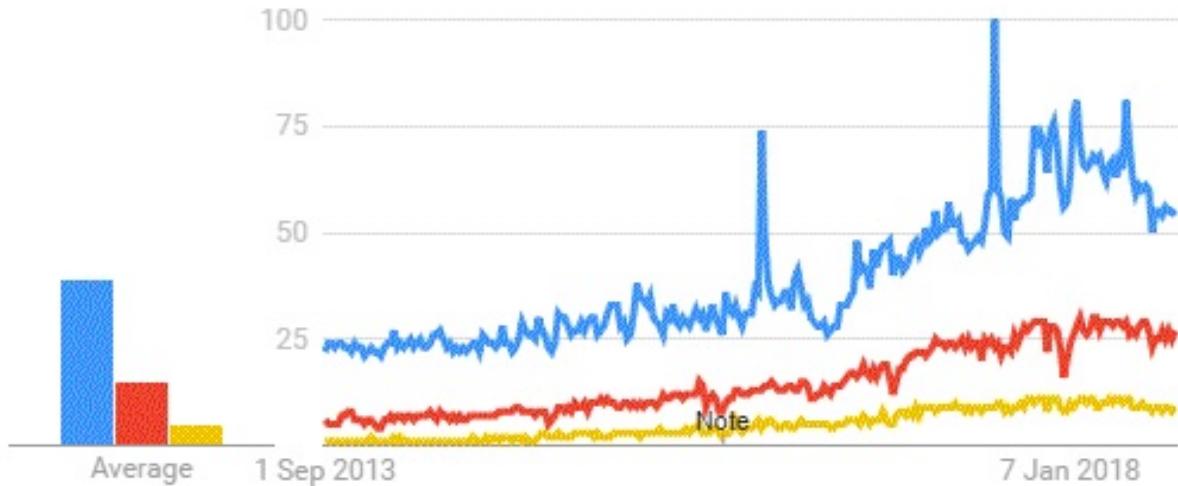
¿Cuándo usar ML o DL?

En la tabla siguiente, resumimos la diferencia entre el aprendizaje automático y el aprendizaje profundo.

	Aprendizaje automático	Aprendizaje profundo
Conjunto de datos de formación	Pequeño	Grande
Elegir características	- Sí. Sí	No
Número de algoritmos	Muchos	Pocos
Tiempo de entrenamiento	Corto	Largo

Con el aprendizaje automático, necesita menos datos para entrenar el algoritmo que el aprendizaje profundo. El aprendizaje profundo requiere un conjunto amplio y diverso de datos para identificar la estructura subyacente. Además, el aprendizaje automático proporciona un modelo más rápido entrenado. La arquitectura de aprendizaje profundo más avanzada puede tardar entre días y una semana en entrenarse. La ventaja del aprendizaje profundo sobre el aprendizaje automático es que es altamente preciso. No es necesario comprender qué entidades son la mejor representación de los datos; la red neuronal aprendió a seleccionar entidades críticas. En el aprendizaje automático, debe elegir por sí mismo qué características incluir en el modelo.

● Artificial intelligence ● machine learning ● deep learning



Resumen

La inteligencia artificial está impartiendo una capacidad cognitiva a una máquina. Los primeros sistemas de IA utilizaron la coincidencia de patrones y sistemas expertos.

La idea detrás del aprendizaje automático es que la máquina puede aprender sin intervención humana. La máquina necesita encontrar una manera de aprender a resolver una tarea dados los datos.

El aprendizaje profundo es el avance en el campo de la inteligencia artificial. Cuando hay suficientes datos para entrenar, el aprendizaje profundo logra resultados impresionantes, especialmente para el reconocimiento de imágenes y la traducción de texto. La razón principal es que la extracción de entidades se realiza automáticamente en las diferentes capas de la red.

Capítulo 3: ¿Qué es TensorFlow?

¿Qué es TensorFlow?

Actualmente, la biblioteca de aprendizaje profundo más famosa del mundo es TensorFlow de Google. El producto de Google utiliza el aprendizaje automático en todos sus productos para mejorar el motor de búsqueda, la traducción, los subtítulos de imágenes o las recomendaciones.

Para dar un ejemplo concreto, los usuarios de Google pueden experimentar una búsqueda más rápida y más refinada con IA. Si el usuario escribe una palabra clave a la barra de búsqueda, Google proporciona una recomendación sobre cuál podría ser la siguiente palabra.



Google quiere utilizar el aprendizaje automático para aprovechar sus conjuntos de datos masivos para ofrecer a los usuarios la mejor experiencia. Tres grupos diferentes utilizan el aprendizaje automático:

- Investigadores
- Científicos de datos
- Programadores.

Todos pueden utilizar el mismo conjunto de herramientas para colaborar

entre sí y mejorar su eficiencia.

Google no solo tiene datos; tienen la computadora más masiva del mundo, por lo que TensorFlow se construyó para escalar. TensorFlow es una biblioteca desarrollada por el equipo de Google Brain para acelerar el aprendizaje automático y la investigación de redes neuronales profundas.

Fue construido para funcionar en varias CPU o GPU e incluso sistemas operativos móviles, y tiene varios envoltorios en varios idiomas como Python, C++ o Java.

Historia de TensorFlow

Hace un par de años, el aprendizaje profundo comenzó a superar a todos los demás algoritmos de aprendizaje automático al dar una gran cantidad de datos. Google vio que podría usar estas redes neuronales profundas para mejorar sus servicios:

- Gmail
- Foto
- Motor de búsqueda de Google

Construyen un marco llamado **Tensorflow** para permitir que investigadores y desarrolladores trabajen juntos en un modelo de IA. Una vez desarrollado y escalado, permite que muchas personas lo usen.

Se hizo público por primera vez a finales de 2015, mientras que la primera versión estable apareció en 2017. Es de código abierto bajo licencia Apache Open Source. Puedes usarlo, modificarlo y redistribuir la versión modificada por una tarifa sin pagar nada a Google.

Arquitectura de TensorFlow

La arquitectura Tensorflow funciona en tres partes:

- Preprocesamiento de los datos
- Construir el modelo
- Entrenar y estimar el modelo

Se llama Tensorflow porque toma entrada como una matriz multidimensional, también conocida como **tensores**. Puede construir una especie de **diagrama de flujo** de las operaciones (denominadas Graph) que desea realizar en esa entrada. La entrada entra en un extremo, y luego fluye a través de este sistema de múltiples operaciones y sale el otro extremo como salida.

Es por eso que se llama TensorFlow porque el tensor entra en él fluye a través de una lista de operaciones, y luego sale por el otro lado.

¿Dónde puede correr Tensorflow?

TensorFlow puede hardware y requisitos de software se pueden clasificar en

Fase de desarrollo: Esto es cuando entrenas el modo. Por lo general, la capacitación se realiza en su escritorio o portátil.

Fase de ejecución o fase de inferencia: Una vez realizado el entrenamiento, Tensorflow se puede ejecutar en muchas plataformas diferentes. Puedes ejecutarlo en

- Escritorio con Windows, macOS o Linux
- Cloud como servicio web
- Dispositivos móviles como iOS y Android

Puede entrenarlo en varias máquinas y luego puede ejecutarlo en una máquina diferente, una vez que tenga el modelo entrenado.

El modelo puede ser entrenado y utilizado en GPU, así como en CPU. Las GPU fueron diseñadas inicialmente para videojuegos. A finales de 2010, los investigadores de Stanford encontraron que GPU también era muy bueno en operaciones de matriz y álgebra por lo que los hace muy rápido para hacer este tipo de cálculos. El aprendizaje profundo se basa en una gran cantidad de multiplicación matricial. TensorFlow es muy rápido para calcular la multiplicación de la matriz porque está escrito en C ++. Aunque está implementado en C ++, TensorFlow puede ser accedido y controlado por otros lenguajes principalmente, Python.

Por último, una característica importante de TensorFlow es el TensorBoard. El TensorBoard permite monitorear gráfica y visualmente lo que TensorFlow está haciendo.

Introducción a los componentes de TensorFlow

Tensor

El nombre de Tensorflow se deriva directamente de su marco principal: **Tensor**. En Tensorflow, todos los cálculos involucran tensores. **vector** o **matriz** de n dimensiones que representa todos los tipos de datos. Todos los valores de un tensor tienen un tipo de datos idéntico con una forma conocida (o parcialmente conocida). La forma de los datos es la dimensionalidad de la matriz o matriz.

Un tensor se puede originar a partir de los datos de entrada o el resultado de un cálculo. En TensorFlow, todas las operaciones se llevan a cabo dentro de un **gráfico**. El gráfico es un conjunto de cálculos que se lleva a cabo sucesivamente. **nodo op** y están conectados entre sí.

El gráfico describe las operaciones y conexiones entre los nodos. Sin embargo, no muestra los valores. El borde de los nodos es el tensor, es decir, una forma de llenar la operación con datos.

Gráficos

TensorFlow hace uso de un marco gráfico. El gráfico recoge y describe todos los cálculos de serie realizados durante el entrenamiento. El gráfico tiene muchas ventajas:

- Se hizo para funcionar en varias CPU o GPU e incluso en el sistema operativo móvil
- La portabilidad del gráfico permite conservar los cálculos para su uso inmediato o posterior. El gráfico se puede guardar para ser ejecutado en el futuro.

- Todos los cálculos en el gráfico se realizan conectando tensores juntos
 - Un tensor tiene un nodo y un borde. El nodo lleva la operación matemática y produce una salida de puntos finales. Los bordes de los bordes explican las relaciones de entrada/salida entre los nodos.

¿Por qué TensorFlow es popular?

TensorFlow es la mejor biblioteca de todos porque está construida para ser accesible para todos. La biblioteca Tensorflow incorpora diferentes API para construir a escala arquitectura de aprendizaje profundo como CNN o RNN. TensorFlow se basa en el cálculo gráfico; permite al desarrollador visualizar la construcción de la red neuronal con Tensorboard. Esta herramienta es útil para depurar el programa. Finalmente, Tensorflow está construido para ser implementado a escala. Se ejecuta en CPU y GPU.

Tensorflow atrae la mayor popularidad en GitHub en comparación con el otro marco de aprendizaje profundo.

Lista de algoritmos prominentes soportados por TensorFlow

Actualmente, TensorFlow 1.10 tiene una API incorporada para:

- Regresión lineal: `TF.estimator.Lineal Regresisor`
- Clasificación: `TF.estimator.LinearClassifier`
- Clasificación de aprendizaje profundo: `tf.estimator.dnnClassifier`
- Deep learning wipre and deep:
`tf.estimator.dnnlinearCombinedClassifier`
- Regresión del árbol de refuerzo: `tf.estimator.boostedTreesRegressor`
- Clasificación de árbol potenciada:
`tf.estimator.boostedTreesClassifier`

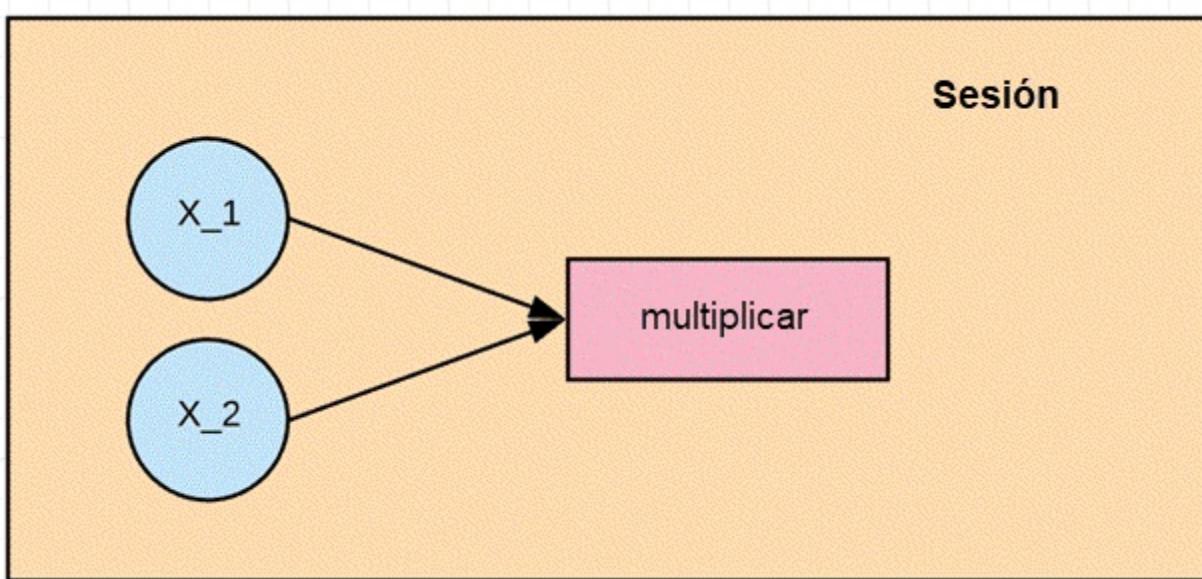
Ejemplo Simple de TensorFlow

```
import numpy as np  
import tensorflow as tf
```

En las dos primeras líneas de código, hemos importado tensorflow como tf. Con Python, es una práctica común usar un nombre corto para una biblioteca. La ventaja es evitar escribir el nombre completo de la biblioteca cuando necesitamos usarlo. Por ejemplo, podemos importar tensorflow como tf, y llamar a tf cuando queremos usar una función tensorflow

Vamos a practicar el flujo de trabajo elemental de Tensorflow con un ejemplo simple. Vamos a crear un gráfico computacional que multiplica dos números juntos.

Durante el ejemplo, multiplicaremos X_1 y X_2 juntos. Tensorflow creará un nodo para conectar la operación. En nuestro ejemplo, se llama multiplicar. Cuando se determina el gráfico, los motores computacionales de Tensorflow se multiplicarán juntos X_1 y X_2.



Finalmente, ejecutaremos una sesión de TensorFlow que ejecutará el gráfico computacional con los valores de X_1 y X_2 e imprimirá el resultado de la multiplicación.

Vamos a definir los nodos de entrada X_1 y X_2. Cuando creamos un nodo en Tensorflow, tenemos que elegir qué tipo de nodo crear. Los nodos X1 y X2 serán un nodo marcador de posición. El marcador de posición asigna un nuevo valor cada vez que hacemos un cálculo. Los crearemos como un nodo de marcador de posición de puntos TF.

Paso 1: Definir la variable

```
x_1 = tf.placeholder(tf.float32, name = "X_1")
x_2 = tf.placeholder(tf.float32, name = "X_2")
```

Cuando creamos un nodo marcador de posición, tenemos que pasar el tipo de datos va a agregar números aquí para que podamos usar un tipo de datos de punto flotante, vamos a usar tf.float32. También necesitamos darle un nombre a este nodo. Este nombre aparecerá cuando veamos las visualizaciones gráficas de nuestro modelo. Vamos a nombrar este nodo X_1 pasando un parámetro llamado nombre con un valor de X_1 y ahora vamos a definir X_2 de la misma manera. X_2.

Paso 2: Definir el cálculo

```
multiply = tf.multiply(x_1, x_2, name = "multiply")
```

Ahora podemos definir el nodo que hace la operación de multiplicación. En Tensorflow podemos hacer eso creando un nodo tf.multiply.

Pasaremos los nodos X_1 y X_2 al nodo de multiplicación. Le dice a tensorflow que vincule esos nodos en el gráfico computacional, por lo que le pedimos que extraiga los valores de x e y multiplique el resultado. También vamos a dar al nodo de multiplicación el nombre de multiplicar. Es la definición completa de nuestro gráfico computacional simple.

Paso 3: Ejecutar la operación

Para ejecutar operaciones en el gráfico, tenemos que crear una sesión. En Tensorflow, lo hace `TF.session()`. Ahora que tenemos una sesión podemos pedirle a la sesión que ejecute operaciones en nuestro gráfico computacional llamando a `sesión`. Para ejecutar el cálculo, necesitamos usar `run`.

Cuando se ejecute la operación de adición, verá que necesita tomar los valores de los nodos `X_1` y `X_2`, por lo que también necesitamos alimentar valores para `X_1` y `X_2`. Podemos hacerlo suministrando un parámetro llamado `feed_dict`. Pasamos el valor `1,2,3` para `X_1` y `4,5,6` para `X_2`.

Imprimimos los resultados con `print(resultado)` Deberíamos ver `4, 10 y 18` para `1x4, 2x5` y `3x6`

```
x_1 = tf.placeholder(tf.float32, name = "X_1")
x_2 = tf.placeholder(tf.float32, name = "X_2")

multiply = tf.multiply(x_1, x_2, name = "multiply")

with tf.Session() as session:
    result = session.run(multiply, feed_dict={x_1:[1,2,3], x_2:
[4,5,6]})
    print(result)
```

[4. 10. 18.]

Opciones para cargar datos en TensorFlow

El primer paso antes de entrenar un algoritmo de aprendizaje automático es cargar los datos. Hay dos formas comunes de cargar datos:

1. Cargar datos en la memoria: Es el método más simple. Cargan todos sus datos en la memoria como una sola matriz. Puede escribir un código de Python. Estas líneas de código no están relacionadas con Tensorflow.
2. Tubería de datos de Tensorflow. Tensorflow tiene API incorporada que le ayuda a cargar los datos, realizar la operación y alimentar el algoritmo de aprendizaje automático fácilmente. Este método funciona muy bien, especialmente cuando tiene un conjunto de datos grande. Por ejemplo, se sabe que los registros de imágenes son enormes y no encajan en la memoria. La canalización de datos administra la memoria por sí misma

¿Qué solución usar?

Cargar datos en memoria

Si su conjunto de datos no es demasiado grande, es decir, menos de 10 gigabytes, puede usar el primer método. Los datos pueden caber en la memoria. Puede utilizar una famosa biblioteca llamada Pandas para importar archivos CSV. Usted aprenderá más acerca de los pandas en el próximo tutorial.

Cargar datos con tubería Tensorflow

El segundo método funciona mejor si tiene un conjunto de datos grande. Por ejemplo, si tiene un conjunto de datos de 50 gigabytes y su computadora tiene sólo 16 gigabytes de memoria, entonces la máquina se

bloqueará.

En esta situación, debe construir una tubería Tensorflow. La canalización cargará los datos en lotes o en trozos pequeños. Cada lote será empujado a la tubería y estará listo para el entrenamiento. Construir una tubería es una solución excelente porque le permite usar computación paralela. Significa que Tensorflow entrenará el modelo a través de varias CPU. Fomenta el cálculo y permite el entrenamiento de red neuronal potente.

Verá en los próximos tutoriales sobre cómo construir una tubería significativa para alimentar su red neuronal.

En pocas palabras, si tiene un conjunto de datos pequeño, puede cargar los datos en memoria con la biblioteca Pandas.

Si tiene un conjunto de datos grande y desea hacer uso de varias CPU, entonces será más cómodo trabajar con la canalización Tensorflow.

Crear tubería Tensorflow

En el ejemplo anterior, agregamos manualmente tres valores para X_1 y X_2. Ahora veremos cómo cargar datos en Tensorflow.

Paso 1) Crear los datos

En primer lugar, usemos la biblioteca numpy para generar dos valores aleatorios.

```
import numpy as np  
x_input = np.random.sample((1,2))  
print(x_input)
```

[[0.8835775 0.23766977]]

Paso 2: Crear el marcador de posición

Al igual que en el ejemplo anterior, creamos un marcador de posición con el nombre X. Necesitamos especificar explícitamente la forma del tensor. En caso de que, cargaremos una matriz con solo dos valores. Podemos escribir la forma como forma = [1,2]

```
# using a placeholder  
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

Paso 3: Definir el método de dataset

a continuación, tenemos que definir el conjunto de datos donde podemos llenar el valor del marcador de posición x. Necesitamos utilizar el método `tf.data.Dataset.from_tensor_slices`

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

Paso 4: Crear la tubería

En el paso cuatro, tenemos que inicializar la tubería donde fluirán los datos. Necesitamos crear un iterador con `make_initializable_iterator`. Lo nombramos iterador. Luego tenemos que llamar a este iterador para alimentar el siguiente lote de datos, `get_next`. Nós nomeamos este passo `get_next`. Tenga en cuenta que en nuestro ejemplo, solo hay un lote de datos con solo dos valores.

```
iterator = dataset.make_initializable_iterator()  
get_next = iterator.get_next()
```

Paso 5: Ejecutar la operación

El último paso es similar al ejemplo anterior. Iniciamos una sesión y ejecutamos el iterador de operación. Alimentamos el `feed_dict` con el valor generado por numpy. Estos dos valores llenarán el marcador de posición `x`. Luego ejecutamos `get_next` para imprimir el resultado.

```
with tf.Session() as sess:  
    # feed the placeholder with data  
    sess.run(iterator.initializer, feed_dict={ x: x_input })  
    print(sess.run(get_next)) # output [ 0.52374458  0.71968478]
```

```
[0.8835775  0.23766978]
```

Resumen

TensorFlow es la biblioteca de aprendizaje profundo más famosa en estos últimos años. Un practicante que usa TensorFlow puede construir cualquier estructura de aprendizaje profundo, como CNN, RNN o simple red neuronal artificial.

TensorFlow es utilizado principalmente por académicos, startups y grandes empresas. Google utiliza TensorFlow en casi todos los productos diarios de Google, incluidos Gmail, Photo y Google Search Engine.

El equipo de Google Brain desarrolló TensorFlow para llenar la brecha

entre investigadores y desarrolladores de productos. En 2015, hicieron público TensorFlow; está creciendo rápidamente en popularidad. Hoy en día, TensorFlow es la biblioteca de aprendizaje profundo con más repositorios en GitHub.

Los profesionales usan Tensorflow porque es fácil de implementar a escala. Está diseñado para funcionar en la nube o en dispositivos móviles como iOS y Android.

Tensorflow funciona en una sesión. Cada sesión se define mediante un gráfico con diferentes cálculos. Un ejemplo simple puede ser multiplicar a número. En Tensorflow, se requieren tres pasos:

1. Definir la variable

```
x_1 = tf.placeholder(tf.float32, name = "X_1")
x_2 = tf.placeholder(tf.float32, name = "X_2")
```

2. Definir el cálculo

```
multiply = tf.multiply(x_1, x_2, name = "multiply")
```

3. Ejecutar la operación

```
with tf.Session() as session:
    result = session.run(multiply, feed_dict={x_1:[1,2,3], x_2:[4,5,6]})
    print(result)
```

Una práctica común en Tensorflow es crear una canalización para cargar los datos. Si sigues estos cinco pasos, podrás cargar datos en TensorFlow

1. Crear los datos

```
import numpy as np
x_input = np.random.sample((1,2))
print(x_input)
```

2. Crear el marcador de posición

```
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

3. Definir el método de dataset

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

4. Crear la tubería

```
iterator = dataset.make_initializable_iterator() get_next =  
iterator.get_next()
```

5. Ejecutar el programa

```
with tf.Session() as sess:  
sess.run(iterator.initializer, feed_dict={ x: x_input })  
print(sess.run(get_next))
```

Capítulo 4: Comparación de bibliotecas de aprendizaje profundo

La inteligencia artificial está creciendo en popularidad desde 2016 con, el 20% de las grandes empresas que utilizan IA en sus negocios (informe McKinsey, 2018). Según el mismo informe, AI puede crear un valor sustancial en todas las industrias. En la banca, por ejemplo, el potencial de la IA se estima en \$300 mil millones, en el comercio minorista el número se dispara a \$600 mil millones.

Para desbloquear el valor potencial de la IA, las empresas deben elegir el marco de aprendizaje profundo adecuado. En este tutorial, aprenderá acerca de las diferentes bibliotecas disponibles para llevar a cabo tareas de aprendizaje profundo. Algunas bibliotecas han existido durante años, mientras que una nueva biblioteca como TensorFlow ha salido a la luz en los últimos años.

8 Mejores Bibliotecas/Marco de aprendizaje profundo

En esta lista, compararemos los principales marcos de aprendizaje profundo. Todos ellos son de código abierto y populares en la comunidad científica de datos. También vamos a comparar ML populares como proveedores de servicios

Antorcha

Torch es una antigua biblioteca de aprendizaje automático de código abierto. Fue lanzado por primera vez hace 15 años. Es lenguajes de programación primarios es LUA, pero tiene una implementación en C. Torch soporta una vasta biblioteca para algoritmos de aprendizaje automático, incluyendo el aprendizaje profundo. Es compatible con la implementación CUDA para el cálculo paralelo.

La antorcha es utilizada por la mayoría de los laboratorios líderes como Facebook, Google, Twitter, Nvidia, y así sucesivamente. Torch tiene una biblioteca en Python nombres Pytorch.

Infer.net

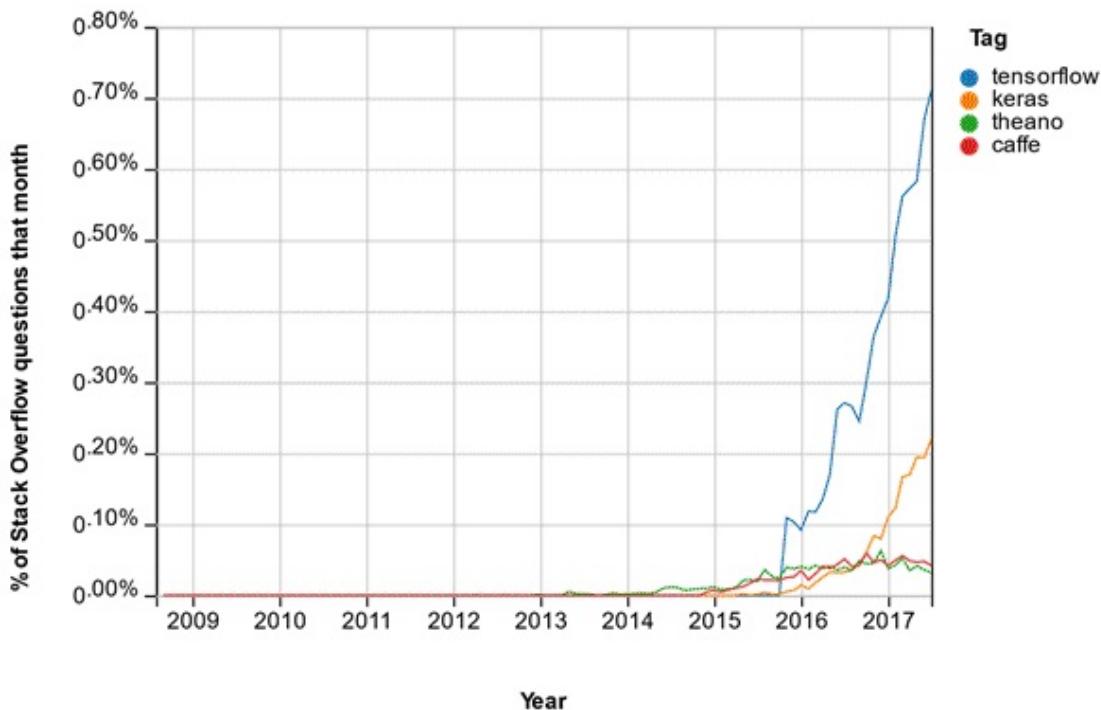
Infer.net es desarrollado y mantenido por Microsoft. Infer.net es una biblioteca con un enfoque primario en la estadística bayesiana. Infer.net está diseñado para ofrecer a los profesionales algoritmos de vanguardia para el modelado probabilístico. A biblioteca contém ferramentas analíticas como análise bayesiana, cadeia Markov escondida, clustering.

Keras

Keras es un marco de Python para el aprendizaje profundo. Es una

biblioteca conveniente para construir cualquier algoritmo de aprendizaje profundo. La ventaja de Keras es que usa el mismo código Python para ejecutarse en CPU o GPU. Además, el entorno de codificación es puro y permite entrenar algoritmo de última generación para visión por ordenador, reconocimiento de texto entre otros.

Keras ha sido desarrollado por François Chollet, investigador de Google. Keras se utiliza en organizaciones prominentes como CERN, Yelp, Square o Google, Netflix y Uber.



Theanó

Theanos es una biblioteca de aprendizaje profundo desarrollada por la Universidad de Montreal en 2007. Ofrece computación rápida y se puede ejecutar tanto en CPU como en GPU. Theanos ha sido desarrollado para entrenar algoritmos de redes neuronales profundas.

MICROSOFT COGNITIVE TOOLKIT (CNTK)

Microsoft Toolkit, anteriormente conocido como CNTK, es una biblioteca de aprendizaje profundo desarrollada por Microsoft. Según Microsoft, la biblioteca es una de las más rápidas del mercado. Microsoft Toolkit es una biblioteca de código abierto, aunque Microsoft la está utilizando extensamente para su producto como Skype, Cortana, Bing y Xbox. El kit de herramientas está disponible tanto en Python como en C + +.

MXNet

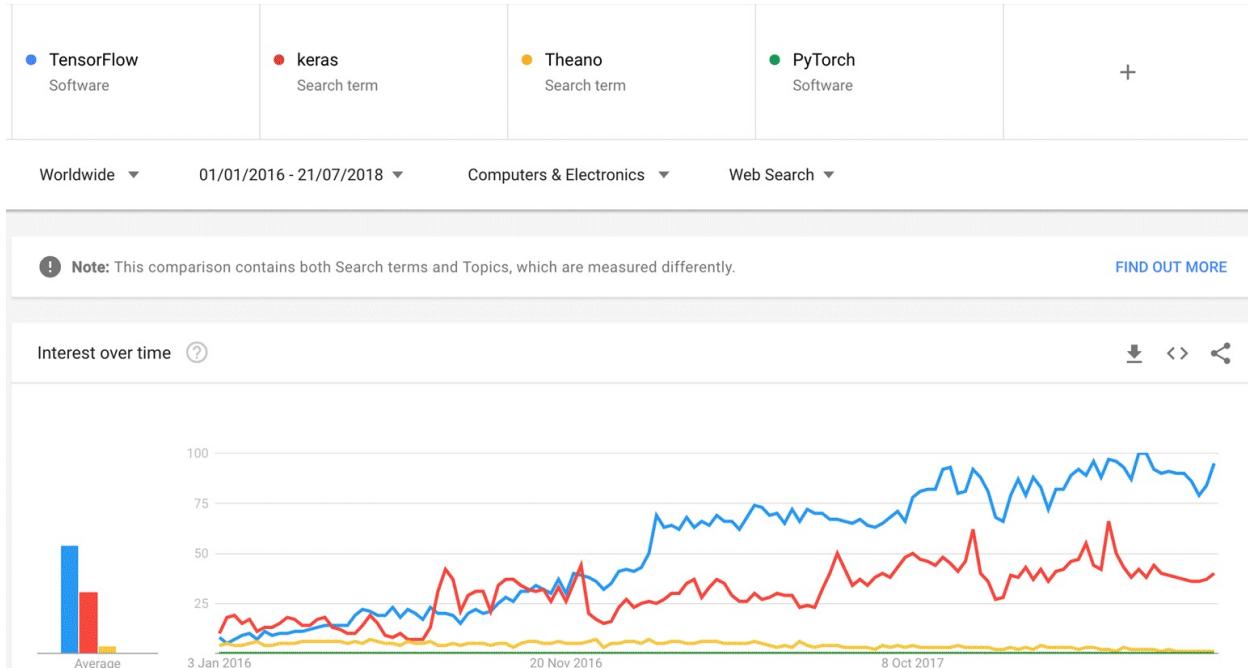
MxNet es una biblioteca de aprendizaje profundo reciente. Es accesible con múltiples lenguajes de programación incluyendo C + +, Julia, Python y R. MxNet se puede configurar para trabajar tanto en CPU como en GPU. MxNet incluye una arquitectura de aprendizaje profundo de última generación, como la red neuronal convolucional y la memoria a corto plazo. MxNet está construido para trabajar en armonía con la infraestructura de nube dinámica. El usuario principal de MxNet es Amazon

Caffe

Caffe es una biblioteca construida por Yangqing Jia cuando era estudiante de doctorado en Berkeley. Caffe está escrito en C + + y puede realizar cálculos tanto en CPU como en GPU. Los principales usos de Caffe es la red neuronal convolucional. Aunque, En 2017, Facebook extendió Caffe con arquitectura de aprendizaje más profundo, incluyendo red neuronal recurrente. Caffe es utilizado por académicos y startups, pero también algunas grandes empresas como Yahoo!.

TensorFlow

TensorFlow es un proyecto de código abierto de Google. TensorFlow es la biblioteca de aprendizaje profundo más famosa en estos días. Fue lanzado al público a finales de 2015



TensorFlow se desarrolla en C++ y tiene una API de Python conveniente, aunque las API de C++ también están disponibles. Empresas prominentes como Airbus, Google, IBM, etc. están utilizando TensorFlow para producir algoritmos de aprendizaje profundo.

TensorFlow vs Theanos vs Torch vs Keras vs infer.net vs CNTK vs MXNet vs Caffe: Diferencias clave

Biblioteca	Plataforma	Escrito en	Soporte Cuda	Ejecución paralela	Ha entrenado modelos	RNN	CNN
Antorcha	Linux, macOS, Windows	Lua	Yes	Yes	Yes	Yes	Yes
Infer.Net	Linux, macOS, Windows	Estudio Visual	No	No	No	No	No
Keras	Linux, macOS, Windows	Python	Yes	Yes	Yes	Yes	Yes
Theano	Multiplataforma	Python	Yes	Yes	Yes	Yes	Yes
TensorFlow	Linux, macOS, Windows, Android	C++, Python, CUDA	Yes	Yes	Yes	Yes	Yes
MICROSOFT COGNITIVE TOOLKIT	Linux, Windows, Mac con Docker	C++	Yes	Yes	Yes	Yes	Yes
Caffe	Linux, macOS, Windows	C++	Yes	Yes	Yes	Yes	Yes
MXNet	Linux, Windows, macOS, Android, iOS, Javascript	C++	Yes	Yes	Yes	Yes	Yes

Veredicto:

TensorFlow es la mejor biblioteca de todos porque está construida para ser accesible para todos. La biblioteca Tensorflow incorpora diferentes API para construir a escala arquitectura de aprendizaje profundo como CNN o RNN. TensorFlow se basa en el cálculo gráfico, que permite al desarrollador visualizar la construcción de la red neuronal con

Tensorboard. Esta herramienta es útil para depurar el programa. Finalmente, Tensorflow está construido para ser implementado a escala. Se ejecuta en CPU y GPU.

Tensorflow atrae la mayor popularidad en GitHub en comparación con el otro marco de aprendizaje profundo.

Comparación del aprendizaje automático como servicio

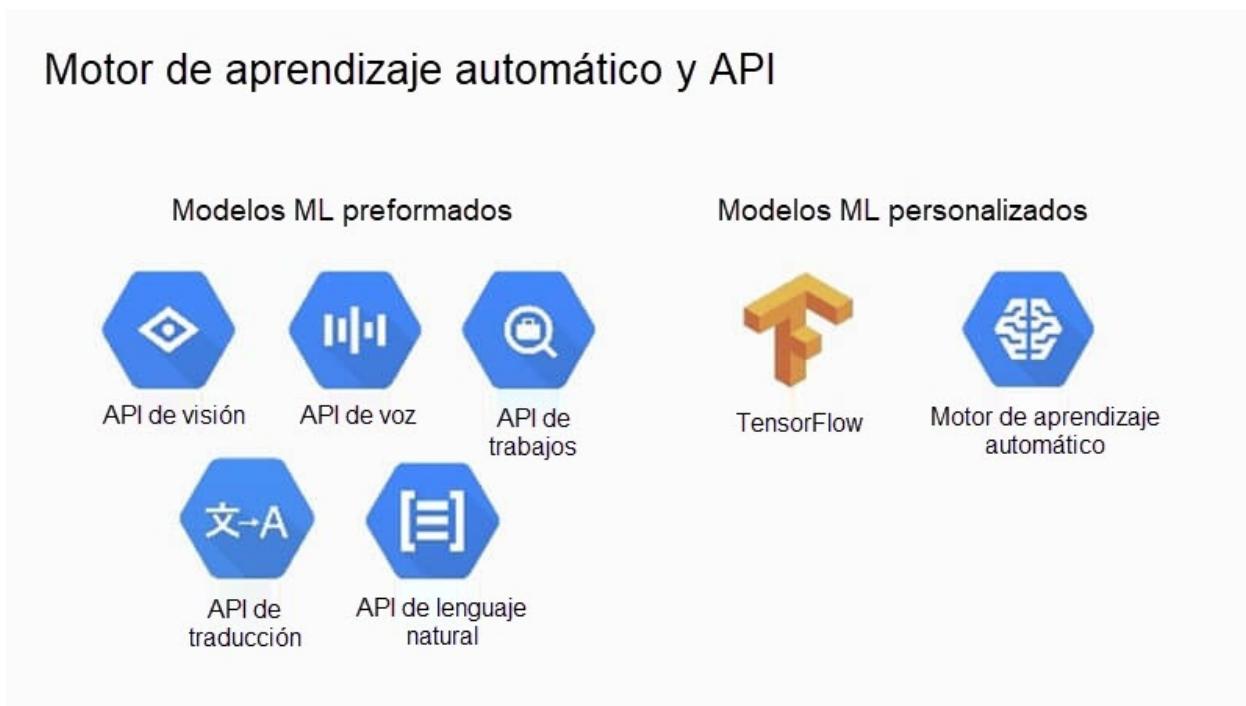
Los siguientes son 4 DL populares como proveedores de servicios

Google Cloud ML

Google proporciona un modelo preentrenado para desarrolladores disponible en Cloud Automl. Esta solución existe para un desarrollador sin una sólida experiencia en aprendizaje automático. Los desarrolladores pueden utilizar el modelo preentrenado de Google de última generación en sus datos. Permite a los desarrolladores entrenar y evaluar cualquier modelo en pocos minutos.

Google ofrece actualmente una API REST para visión por computadora, reconocimiento de voz, traducción y PNL.

Motor de aprendizaje automático y API



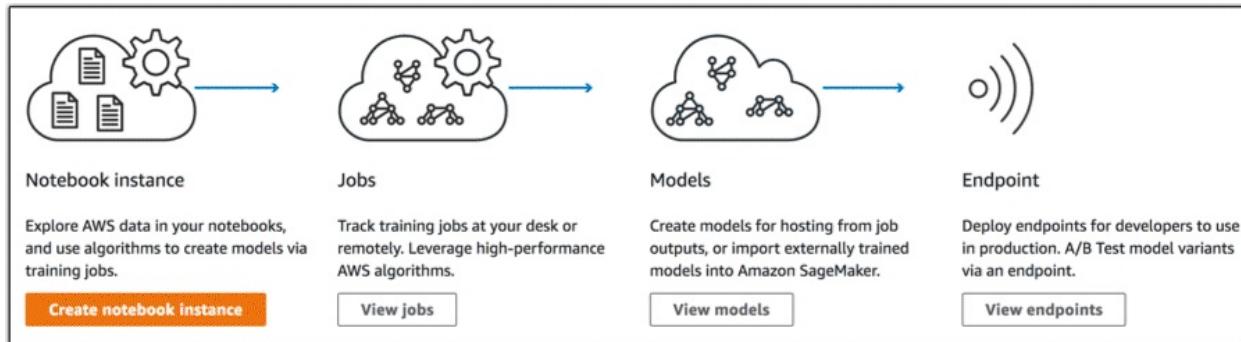
Utilizando Google Cloud, puedes entrenar un marco de aprendizaje automático basado en TensorFlow, SciKit-Learn, XGBoost o Keras. El aprendizaje automático de Google Cloud capacitará a los modelos en toda su nube.

La ventaja de utilizar la informática en la nube de Google es la simplicidad de implementar el aprendizaje automático en la producción. No es necesario configurar el contenedor Docker. Además, la nube se encarga de la infraestructura. Sabe cómo asignar recursos con CPU, GPU y CPU. Hace que el entrenamiento sea más rápido con cómputos paralelos.

SAGEMaker de AWS

Uno de los principales competidores de Google Cloud es Amazon Cloud, AWS. Amazon ha desarrollado Amazon SAGEMaker para permitir a los científicos y desarrolladores de datos crear, capacitar y poner en producción cualquier modelo de aprendizaje automático.

SageMaker está disponible en un Jupyter Notebook e incluye la biblioteca de aprendizaje automático más utilizada, TensorFlow, MxNet, SciKit-learn entre otros. Los programas escritos con SAGEMaker se ejecutan automáticamente en los contenedores Docker. Amazon gestiona la asignación de recursos para optimizar la formación y la implementación.



Amazon proporciona API a los desarrolladores para agregar inteligencia a sus aplicaciones. En algunas ocasiones, no hay necesidad de reinventar la rueda construyendo desde cero nuevos modelos, mientras que hay potentes modelos preentrenados en la nube. Amazon ofrece servicios API para visión informática, chatbots conversacionales y servicios de idiomas:

Las tres principales API disponibles son:

- Amazon Rekognition: proporciona reconocimiento de imagen y vídeo a una aplicación
- Amazon Comprehend: realizar minería de texto y procesamiento del lenguaje neural para, por ejemplo, automatizar el proceso de verificación de la legalidad de los documentos financieros
- Amazon Lex: Añadir chatbot a una aplicación

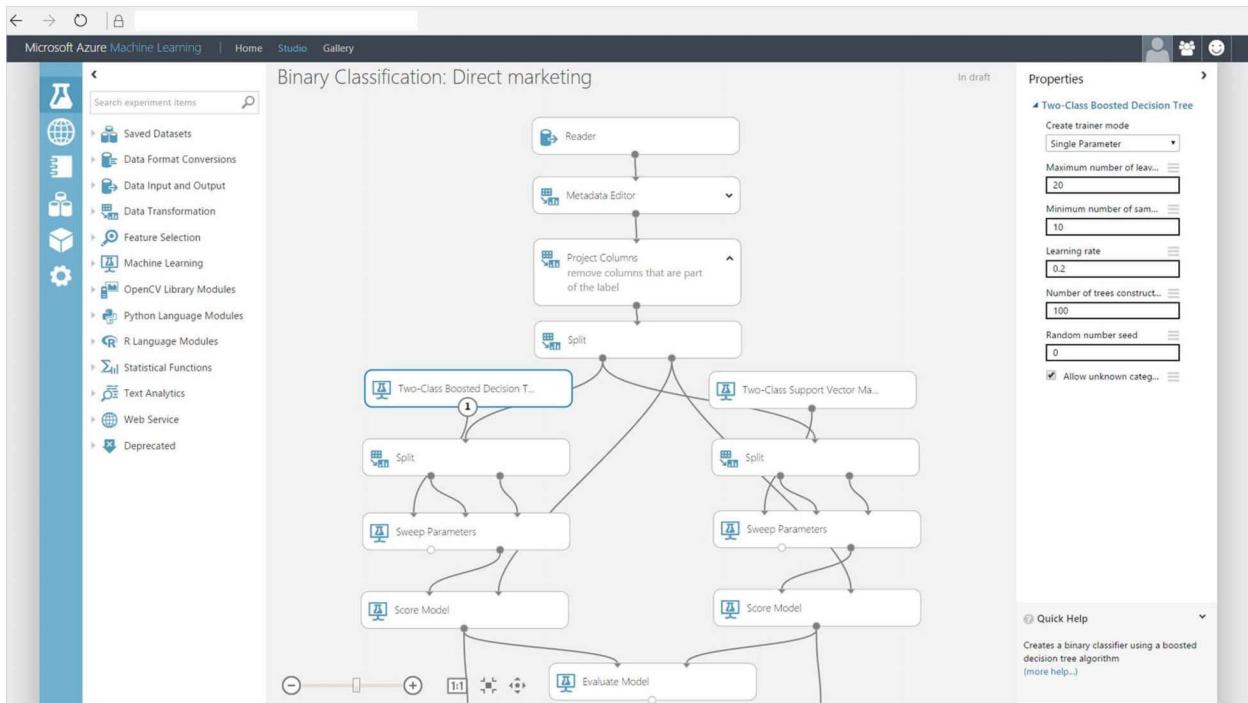
Estudio de aprendizaje automático de Azure

Probablemente uno de los enfoques más amigables para el aprendizaje automático es Azure Machine Learning Studio. La ventaja significativa de

esta solución es que no se requieren conocimientos previos de programación.

Microsoft Azure Machine Learning Studio es una herramienta colaborativa de arrastrar y soltar para crear, capacitar, evaluar e implementar soluciones de aprendizaje automático. El modelo se puede implementar de manera eficiente como servicios web y utilizar en varias aplicaciones como Excel.

La interfaz de aprendizaje automático de Azure es interactiva, lo que permite al usuario crear un modelo simplemente arrastrando y soltando elementos rápidamente.

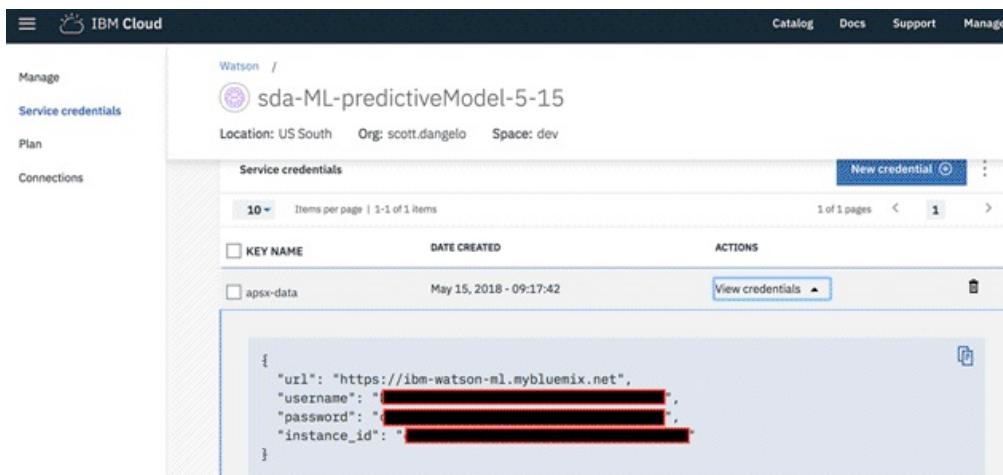


Cuando el modelo esté listo, el desarrollador puede guardarla y enviarla a Azure Gallery o Azure Marketplace.

El aprendizaje automático de Azure se puede integrar en R o Python su paquete integrado personalizado.

IBM Watson ML

Watson studio puede simplificar los proyectos de datos con un proceso optimizado que permite extraer valor y conocimientos de los datos para ayudar al negocio a ser más inteligente y más rápido. Watson studio ofrece un entorno de aprendizaje automático y científico de datos colaborativo fácil de usar para crear y entrenar modelos, preparar y analizar datos y compartir conocimientos en un solo lugar. Watson Studio es fácil de usar con un código de arrastrar y soltar.



The screenshot shows the IBM Cloud Watson interface. On the left, there's a sidebar with 'Manage', 'Service credentials' (which is selected), 'Plan', and 'Connections'. The main area shows a service named 'sda-ML-predictiveModel-5-15' located in 'US South' with 'Org: scott.dangelo' and 'Space: dev'. Under 'Service credentials', there's a table with one item: 'apsx-data' created on 'May 15, 2018 - 09:17:42'. The credential details are shown in a modal window:

```
{  
  "url": "https://ibm-watson-ml.mybluemix.net",  
  "username": "REDACTED",  
  "password": "REDACTED",  
  "instance_id": "REDACTED"  
}
```

Watson studio admite algunos de los marcos más populares como Tensorflow, Keras, Pytorch, Caffe y puede implementar un algoritmo de aprendizaje profundo en las últimas GPU de Nvidia para ayudar a acelerar el modelado.

Veredicto:

En nuestro punto de vista, la solución de nube de Google es la que es la más recomendada. La solución de nube de Google proporciona precios más bajos de AWS en al menos un 30% para almacenamiento de datos y solución de aprendizaje automático. Google está haciendo un excelente trabajo para democratizar la IA. Ha desarrollado un lenguaje de código

abierto, TensorFlow, conexión optimizada de almacenamiento de datos, proporciona herramientas tremendas desde la visualización de datos, el análisis de datos hasta el aprendizaje automático. Además, Google Console es ergonómica y mucho más completa que AWS o Windows.

Capítulo 5: Cómo descargar e instalar TensorFlow Windows y Mac

En este tutorial, vamos a explicar cómo instalar **TensorFlow** con **Anaconda**. Aprenderás a usar TensorFlow con Jupyter. Jupyter es un visor de portátiles.

Versiones de TensorFlow

TensorFlow admite cálculos en varias CPU y GPU. Esto significa que los cálculos se pueden distribuir entre dispositivos para mejorar la velocidad del entrenamiento. Con la paralelización, no es necesario esperar semanas para obtener los resultados de los algoritmos de entrenamiento.

Para el usuario de Windows, TensorFlow proporciona dos versiones:

- **TensorFlow con soporte de CPU solamente:** Si su máquina no se ejecuta en NVIDIA GPU, sólo puede instalar esta versión
- **TensorFlow con compatibilidad con GPU:** Para un cálculo más rápido, puede usar esta versión de TensorFlow. Esta versión tiene sentido solo si necesita una fuerte capacidad computacional.

Durante este tutorial, la versión básica de TensorFlow es suficiente.

Nota: TensorFlow no proporciona compatibilidad con GPU en macOS.

Aquí está cómo proceder

Usuario de macOS:

- Instalar Anaconda
- Cree un archivo.yml para instalar Tensorflow y dependencias
- Lanzamiento de Jupyter Notebook

Para Windows

- Instalar Anaconda
- Crear un archivo.yml para instalar dependencias
- Use pip para agregar TensorFlow
- Lanzamiento de Jupyter Notebook

Para ejecutar Tensorflow con Jupyter, debe crear un entorno dentro de Anaconda. Significa que instalará Ipython, Jupyter y TensorFlow en una carpeta apropiada dentro de nuestra máquina. Además de esto, agregará una biblioteca esencial para la ciencia de datos: “Pandas”. La biblioteca Pandas ayuda a manipular un marco de datos.

Instalar Anaconda

Descargue Anaconda versión 4.3.1 (para Python 3.6) para el sistema apropiado.

Anaconda le ayudará a administrar todas las bibliotecas necesarias ya sea para Python o R. Consulte este tutorial para instalar Anaconda

Crear un .yml para instalar Tensorflow y dependencias

Incluye:

- Localizar el camino de Anaconda
- Establecer el directorio de trabajo en Anaconda
- Cree el archivo yml (para el usuario de macOS, TensorFlow está instalado aquí)
- Editar el archivo yml
- Compilar el archivo yml
- Activar Anaconda
- Instalar TensorFlow (sólo usuario de Windows)

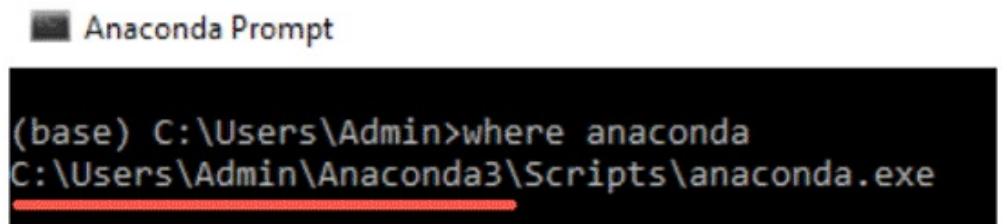
Paso 1) Localice Anaconda

El primer paso que debes hacer es localizar el camino de Anaconda. Creará un nuevo entorno conda que incluye las bibliotecas necesarias que utilizará durante los tutoriales sobre TensorFlow.

Windows

Si usted es un usuario de Windows, puede usar Anaconda Prompt y escribir:

```
C:\>where anaconda
```



The screenshot shows a terminal window titled "Anaconda Prompt". The command "(base) C:\Users\Admin>where anaconda" is entered at the prompt. The output shows the path "C:\Users\Admin\Anaconda3\Scripts\anaconda.exe" highlighted with a red underline, indicating it is the executable file being searched for.

```
(base) C:\Users\Admin>where anaconda
C:\Users\Admin\Anaconda3\Scripts\anaconda.exe
```

Estamos interesados en conocer el nombre de la carpeta donde está instalada Anaconda porque queremos crear nuestro nuevo entorno dentro de esta ruta. Por ejemplo, en la imagen anterior, Anaconda se instala en la carpeta Admin. Para usted, puede lo mismo, es decir, Administrador o el nombre del usuario.

En el siguiente, estableceremos el directorio de trabajo de c:\ to Anaconda3.

MacOS

para el usuario de macOS, puede usar el Terminal y escribir:

```
which anaconda
```

```
Thomass-MacBook-Pro:~ Thomas$ which anaconda  
/Users/Thomas/anaconda3/bin/anaconda
```

Camino de
Anaconda

Tendrá que crear una nueva carpeta dentro de Anaconda que contendrá **Ipitón, Jupyter y TensorFlow**. Una forma rápida de instalar bibliotecas y software es escribir un archivo yml.

Paso 2) Establecer el directorio de trabajo

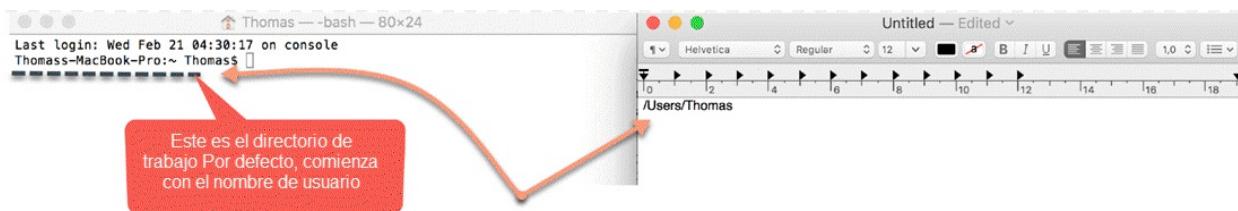
Debe especificar el directorio de trabajo donde desea crear el archivo yml. Como se ha dicho antes, se ubicará dentro de Anaconda.

Para el usuario de macOS:

La Terminal establece el directorio de trabajo predeterminado en

Usuarios/Nombre de usuario. Como puede ver en la figura siguiente, la ruta de anaconda3 y el directorio de trabajo son idénticos. En macOS, la última carpeta se muestra antes de \$. El Terminal instalará todas las bibliotecas de este directorio de trabajo.

Si la ruta del editor de texto no coincide con el directorio de trabajo, puede cambiarla escribiendo cd PATH en el Terminal. PATH es la ruta que ha pegado en el editor de texto. No olvides envolver el PATH con 'PATH'. Esta acción cambiará el directorio de trabajo a PATH.



Abra su terminal y escriba:

```
cd anaconda3
```

Para el usuario de Windows (asegúrese de la carpeta antes de Anaconda3):

```
cd C:\Users\Admin\Anaconda3
```

o el comando path “where anaconda” le da

```
base) C:\Users\Admin>cd C:\Users\Admin\Anaconda3  
base) C:\Users\Admin\Anaconda3>
```

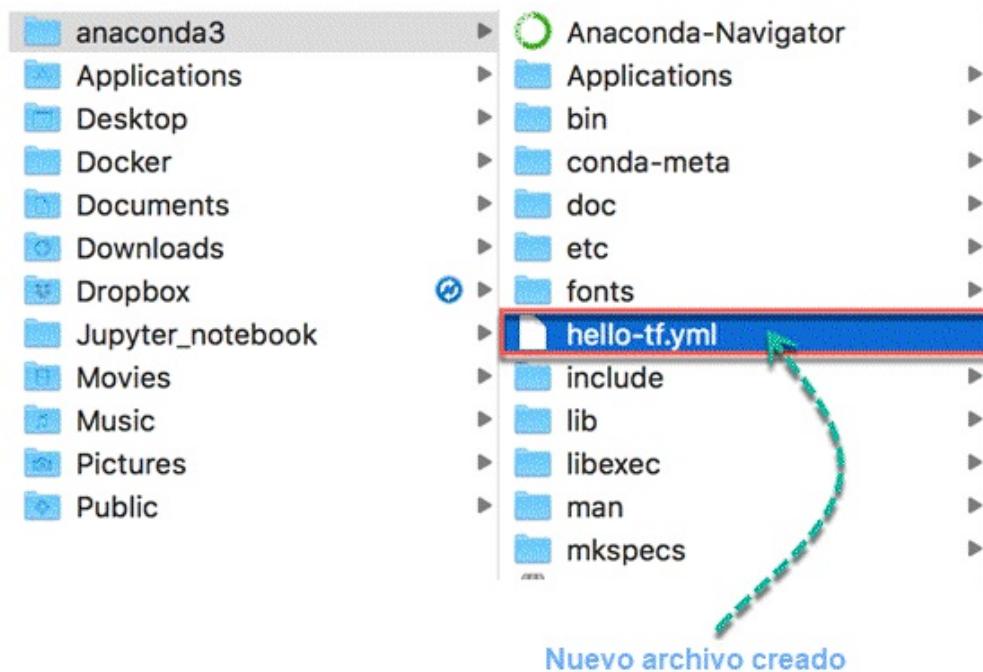
Paso 3) Cree el archivo yml

Puede crear el archivo yml dentro del nuevo directorio de trabajo. El archivo instalará las dependencias que necesita para ejecutar TensorFlow. Copie y pegue este código en la Terminal.

Para el usuario de macOS:

```
touch hello-tf.yml
```

Un nuevo archivo llamado hello-tf.yml debería aparecer dentro de anaconda3



Para el usuario de Windows:

```
echo.>hello-tf.yml
```

Debería aparecer un nuevo archivo llamado hello-tf.yml

This PC > Local Disk (C:) > Users > Admin > Anaconda3 >				
Name	Date modified	Type	Size	
hello-tf.yml	05-04-2018 02:38 ...	YML File	1 KB	
.nonadmin	21-03-2018 12:26 ...	NONADMIN File	0 KB	
Uninstall-Anaconda3.exe	20-02-2018 02:26 ...	Application	296 KB	

Paso 4) Editar el archivo yml

Para el usuario de macOS:

Está listo para editar el archivo yml. Puede pegar el código siguiente en la Terminal para editar el archivo. El usuario de macOS puede usar **vim** para editar el archivo yml.

```
vi hello-tf.yml
```

Hasta ahora, su Terminal se ve así

The terminal session shows the following steps:

1. Establecer ruta: [Thomass-MacBook-Pro:~ Thomas\$ cd anaconda3
2. Crear archivo yml: Thomas\$ touch hello-tf.yml
3. Editar archivo yml: Thomas\$ vi hello-tf.yml

Ingrasa un **editar** Dentro de este modo, puede, después de presionar esc:

- Pulse i para editar
- Presione w para guardar
- Presione q! para salir

Escriba el siguiente código en el modo de edición y pulse esc seguido de:

w

```
name: hello-tf
dependencies:
  - python=3.6
  - jupyter
  - ipython
  - numpy
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.5.0-py3-none-any.whl
```

Nota: El archivo es el caso y intención sensible. Se requieren 2 espacios después de cada intención.

Para macOS

```
name: hello-tf
dependencies:
  - python=3.6
  - jupyter
  - ipython
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl
```

Explicación del código

- name: hello-tf: Nombre del archivo yml
- dependencias:
- python=3.6

- jupyter
- ipython
- pandas: Instalar Python versión 3.6, Jupyter, Ipython y las bibliotecas pandas
- pip: Instalar una biblioteca de Python
 - <https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl>: Instale TensorFlow desde las API de Google.

Presione esc seguido de: q! para todo el modo de edición.

```

name: hello-tf
dependencies:
- python=3.6
- jupyter
- ipython
- numpy
- pandas
- pip:
  - https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.5.0-py3-none-any.whl
~
```

Para el usuario de Windows:

Windows no tiene el programa vim, por lo que el Bloc de notas es suficiente para completar este paso.

```
notepad hello-tf.yml
```

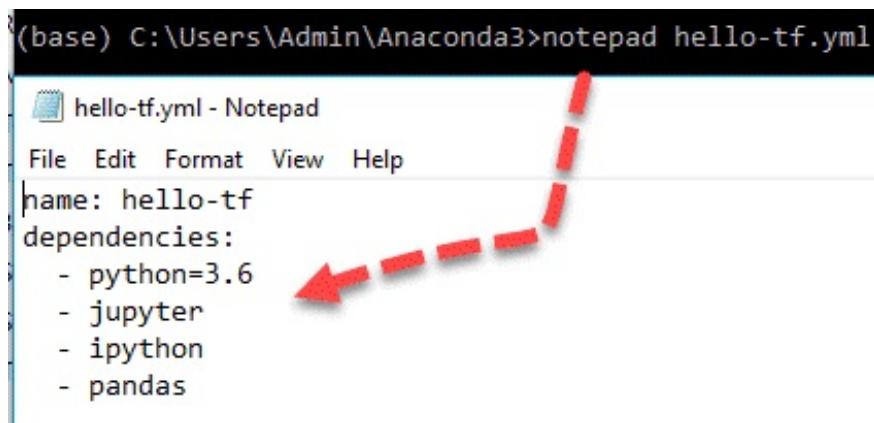
Introduzca el siguiente en el archivo

```
name: hello-tfdependencies:  
- python=3.6  
- jupyter  
- ipython  
- pandas
```

Explicación del código

- name: hello-tf: Nombre del archivo yml
- dependencias:
- python=3.6
- jupyter
- ipython
- pandas: Instalar Python versión 3.6, Jupyter, Ipython y las bibliotecas pandas

Se abrirá el bloc de notas, puede editar el archivo desde aquí.



Nota: Los usuarios de Windows instalarán TensorFlow en el siguiente paso. Neste passo, você apenas preparar o ambiente conda

Paso 5) Compilar el archivo yml

Puede compilar el .yml con el siguiente código:

```
conda env create -f hello-tf.yml
```

Nota: Para los usuarios de Windows, el nuevo entorno se crea dentro del directorio de usuario actual.

Lleva tiempo. Tomará alrededor de 1.1 gb de espacio en el disco duro.

Instalación de bibliotecas



Downloading and Extracting Packages	Progress	Status
python-dateutil 2.7.2: #####	100%	100%
openssl 1.0.2o: #####	100%	100%
imagesize 1.0.0: #####	100%	100%
pytz 2018.3: #####	100%	100%
cffi 1.11.5: #####	100%	100%
tornado 5.0.1: #####	100%	100%
setuptools 39.0.1: #####	100%	100%
jupyter_client 5.2.3: #####	100%	100%
python 3.6.5: #####	100%	100%
zeromq 4.2.3: #####	100%	100%
pexpect 4.4.0: #####	100%	100%
bleach 2.1.3: #####	100%	100%
babel 2.5.3: #####	100%	100%
asn1crypto 0.24.0: #####	100%	100%
libsodium 1.0.16: #####	100%	100%
typing 3.6.4: #####	100%	100%
sphinx 1.7.2: #####	100%	100%
jedi 0.11.1: #####	100%	100%
pip 9.0.3: #####	100%	100%
packaging 17.1: #####	100%	100%
ipywidgets 7.1.2: #####	100%	100%
pysocks 1.6.8: #####	100%	100%
ipykernel 4.8.2: #####	100%	100%
send2trash 1.5.0: #####	100%	100%
pyopenssl 17.5.0: #####	100%	100%
widgetsnbextension 3.1.4: #####	100%	100%
cryptography 2.2.1: #####	100%	100%
pyzmq 17.0.0: #####	100%	100%
notebook 5.4.1: #####	100%	100%

En Windows

```
(base) C:\Users\Admin\Anaconda3>conda env create -f hello-tf.yml
Solving environment: done          .....  
  
==> WARNING: A newer version of conda exists. <==  
    current version: 4.4.10  
    latest version: 4.5.0  
  
Please update conda by running  
  
    $ conda update -n base conda  
  
Downloading and Extracting Packages  
bleach 2.1.3: #####  
numpy 1.14.2: #####  
sip 4.19.8: #####  
jedi 0.11.1: #####  
notebook 5.4.1: #####  
mkl 2018.0.2: #####1
```



Paso 6) Activar entorno conda

Ya casi terminamos. Ahora tiene 2 entornos conda. Ha creado un entorno conda aislado con las bibliotecas que utilizará durante los tutoriales. Esta es una práctica recomendada porque cada proyecto de aprendizaje automático requiere bibliotecas diferentes. Cuando el proyecto haya terminado, puede quitar o no este entorno.

```
conda env list
```

```
Thomass-MACBOOK-Pro:~ Thomas$ conda env list
# conda environment:
#
base          * /Users/Thomas/anaconda3
hello-tf      /Users/Thomas/anaconda3/envs/hello-tf
```

Indique el entorno que se está ejecutando actualmente

entorno principal de la conda

Conda con TensorFlow

El asteríx indica el predeterminado. Debe cambiar a hello-tf para activar el entorno

Para el usuario de macOS:

```
source activate hello-tf
```

Para el usuario de Windows:

```
activate hello-tf
```

```
#  
# To activate this environment, use:  
# > source activate hello-tf  
#  
# To deactivate an active environment, use:  
# > source deactivate  
#
```

Puede comprobar que todas las dependencias están en el mismo entorno. Esto es importante porque permite a Python usar Jupyter y TensorFlow desde el mismo entorno. Si no ves los tres ubicados en la misma carpeta, tienes que volver a empezar.

Para el usuario de macOS:

```
which python  
which jupyter  
which ipython
```

```
Thomass-MacBook-Pro:anaconda3 Thomas$ which python  
/Users/Thomas/anaconda3/bin/python  
Thomass-MacBook-Pro:anaconda3 Thomas$ which jupyter  
/Users/Thomas/anaconda3/bin/jupyter  
Thomass-MacBook-Pro:anaconda3 Thomas$ which ipython  
/Users/Thomas/anaconda3/bin/ipython -
```

Opcional: Puede comprobar si hay actualizaciones.

```
pip install --upgrade tensorflow
```

Paso 7) Sólo para usuarios de Windows: Instalar TensorFlow

Para el usuario de Windows:

```
where python  
where jupyter  
where ipython
```

```
(hello-tf) C:\Users\Admin\Anaconda3>where python  
C:\Users\Admin\Anaconda3\python.exe  
C:\Users\Admin\Anaconda3\envs\hello-tf\python.exe  
  
(hello-tf) C:\Users\Admin\Anaconda3>where jupyter  
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\jupyter.exe  
C:\Users\Admin\Anaconda3\Scripts\jupyter.exe  
  
(hello-tf) C:\Users\Admin\Anaconda3>where ipython  
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\ipython.exe  
C:\Users\Admin\Anaconda3\Scripts\ipython.exe  
  
(hello-tf) C:\Users\Admin\Anaconda3>
```

Como puede ver, ahora tiene dos entornos Python. El principal y el recién creado en i.e. hello-tf. El entorno conda principal no tiene TensorFlow instalado sólo hello-tf. Desde la imagen, python, jupyter e ipython se

instalan en el mismo entorno. Significa que puedes usar TensorFlow con un Jupyter Notebook.

Necesita instalar Tensorflow con el siguiente comando.

```
pip install tensorflow
```

Lanzamiento de Jupyter Notebook

Esta parte es la misma para ambos sistemas operativos.

Puede abrir TensorFlow con Jupyter.

Nota: Cada vez que desee abrir TensorFlow, debe inicializar el entorno
Procederá de la siguiente manera:

- Activar el entorno hello-tf conda
- Abrir Jupyter
- Importar flujo tensorflow
- Eliminar bloc de notas
- Cerrar Jupyter

Paso 1) Activar conda

Para el usuario de macOS:

```
source activate hello-tf
```

Para el usuario de Windows:

```
conda activate hello-tf
```

```
[Thomass-MacBook-Pro:anaconda3 Thomass$ source activate hello-tf  
(hello-tf) Thomass-MacBook-Pro:anaconda3 Thomas$]
```

Paso 2) Abra Jupyter

Después de eso, puede abrir Jupyter desde la Terminal

```
jupyter notebook
```

```

[Thomas-MacBook-Pro:anaconda3 Thomas$ source activate hello-tf
[hello-tf] Thomas-MacBook-Pro:anaconda3 Thomas$ jupyter notebook
[W 16:20:59.106 NotebookApp] Error loading server extension jupyter_tensorboard
  Traceback (most recent call last):
    File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-packages/notebook/notebookapp.py", line 1451, in init_server_extensions
      mod = importlib.import_module(modulename)
    File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/importlib/__init__.py", line 126, in import_module
      return _bootstrap._gcd_import(name[level:], package, level)
    File "<frozen importlib._bootstrap>", line 994, in _gcd_import
    File "<frozen importlib._bootstrap>", line 971, in _find_and_load
    File "<frozen importlib._bootstrap>", line 953, in _find_and_load_unlocked
ModuleNotFoundError: No module named 'jupyter_tensorboard'
[I 16:20:59.113 NotebookApp] Serving notebooks from local directory: /Users/Thomas/anaconda3
[I 16:20:59.113 NotebookApp] 0 active kernels
[I 16:20:59.113 NotebookApp] The Jupyter Notebook is running at:
[I 16:20:59.113 NotebookApp] http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.113 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:20:59.114 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
  http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.378 NotebookApp] Accepting one-time-token-authenticated connection from ::1
[W 16:21:00.612 NotebookApp] 404 GET /api/tensorboard?_=1522570860085 (::1) 24.15ms referer=http://localhost:8888/tree

```

Si el navegador no se abre automáticamente,
copie esta URL.

Su navegador debe abrirse automáticamente, de lo contrario copiar y pegar la url proporcionada por el Terminal. Comienza por <http://localhost:8888>

Dentro del Cuaderno Jupyter, puede ver todos los archivos dentro del directorio de trabajo. Para crear un nuevo cuaderno, simplemente haga clic en **nuevo** y **Python 3**

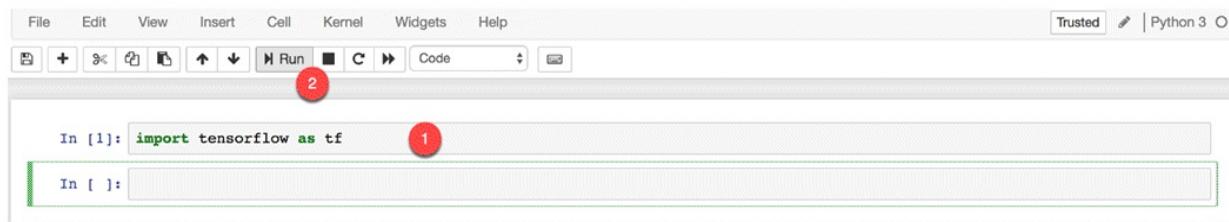
Nota: El nuevo bloc de notas se guarda automáticamente dentro del directorio de trabajo.



Paso 3) Importar Tensorflow

Dentro del cuaderno, puede importar TensorFlow con el alias tf. Haga clic para ejecutar. A continuación se crea una nueva celda.

```
import tensorflow as tf
```



Vamos a escribir su primer código con TensorFlow.

```
hello = tf.constant('Hello, Guru99!')  
hello
```

Se crea un nuevo tensor. Felicitaciones. Usted instala correctamente TensorFlow con Jupyter en su máquina.



Paso 4) Eliminar archivo

Puede eliminar el archivo llamado Untitled.ipynb dentro de Jupyter.

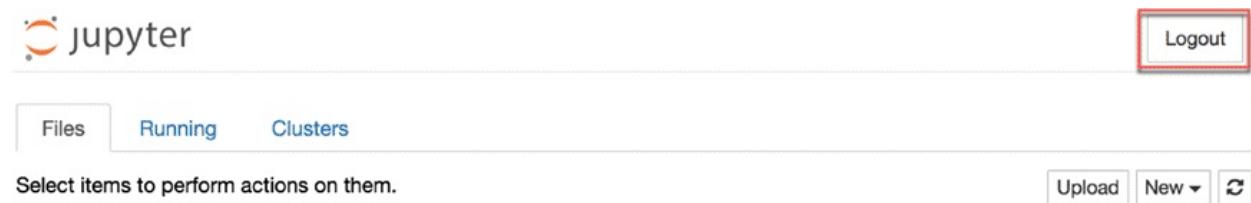


Paso 5) Cerrar Jupyter

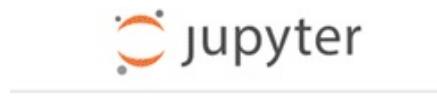
Hay dos formas de cerrar Jupyter. La primera forma es directamente desde el cuaderno. La segunda forma es mediante el uso del terminal (o Anaconda Prompt)

De Jupyter

En el panel principal de Jupyter Notebook, simplemente haga clic en **Cerrar sesión**



Se le redirige a la página de cierre de sesión.



Desde la terminal

Seleccione el terminal o indicador de Anaconda y ejecute dos veces ctr + c.

La primera vez que realice ctr + c, se le pedirá que confirme que desea apagar el portátil. Repita ctr + c para confirmar

```
^C[I 12:38:40.238 NotebookApp] interrupted
Serving notebooks from local directory: /Users/Thomas
1 active kernel
The Jupyter Notebook is running at:
http://localhost:8888/?token=79edffd9c156eac054cf668c6576cb074716c2182a391076
Shutdown this notebook server (y/[n])? No answer for 5s: resuming operation...
```

Use dos veces
Ctr + c para
cerrar Jupyter

```
[I 12:38:57.737 NotebookApp] Shutting down 1 kernel
[I 12:38:58.043 NotebookApp] Kernel shutdown: 2a34c1da-9b2e-40ea-aa20-2971ac33de
1c
(hello-tf) Thomass-MacBook-Pro:~ Thomas$
```

Ha cerrado sesión correctamente.

Jupyter con el entorno principal de conda

Si desea iniciar TensorFlow con jupyter para su uso futuro, debe abrir una nueva sesión con

```
source activate hello-tf
```

Si no lo hace, Jupyter no encontrará TensorFlow

```
In [1]: import tensorflow as tf
-----
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-41389fad42b5> in <module>()
      1 import tensorflow as tf
ModuleNotFoundError: No module named 'tensorflow'
```

Capítulo 6: Tutorial de Cuaderno de Jupyter

¿Qué es Jupyter Notebook?

Un cuaderno Jupyter es una aplicación web que permite al usuario escribir códigos y elementos de texto enriquecido. Dentro de los cuadernos, puede escribir párrafos, ecuaciones, título, agregar enlaces, figuras, etc. Un cuaderno es útil para compartir algoritmos interactivos con su audiencia centrándose en enseñar o demostrar una técnica. Jupyter Notebook es también una forma conveniente de ejecutar el análisis de datos.

Aplicación para portátiles Jupyter

La aplicación Jupyter Notebook es la interfaz donde puedes escribir tus scripts y códigos a través de tu navegador web. La aplicación se puede utilizar localmente, lo que significa que no necesita acceso a Internet o un servidor remoto.



The screenshot shows the Jupyter Notebook interface. At the top, there is a header with a logo, the word "jupyter", and a "Logout" button. Below the header, there are three tabs: "Files" (which is selected), "Running", and "Clusters". A sub-header says "Select items to perform actions on them." On the left, there is a sidebar with a file tree showing the directory structure: a root folder containing "anaconda", "Applications", "Desktop", "Docker", "Documents", "Downloads", "Dropbox", "Jupyter_notebook" (which is currently selected), "Movies", "Music", "Pictures", and "Public". On the right, there is a list of files with their names and last modified times. The list includes "anaconda" (19 hours ago), "Applications" (2 months ago), "Desktop" (9 minutes ago), "Docker" (2 months ago), "Documents" (8 days ago), "Downloads" (an hour ago), "Dropbox" (a day ago), "Jupyter_notebook" (18 minutes ago), "Movies" (4 months ago), "Music" (4 months ago), "Pictures" (2 months ago), and "Public" (8 months ago). At the bottom of the interface, there are buttons for "Upload", "New", and "Create".

Name	Last Modified
anaconda	19 hours ago
Applications	2 months ago
Desktop	9 minutes ago
Docker	2 months ago
Documents	8 days ago
Downloads	an hour ago
Dropbox	a day ago
Jupyter_notebook	18 minutes ago
Movies	4 months ago
Music	4 months ago
Pictures	2 months ago
Public	8 months ago

Cada cálculo se realiza a través de un núcleo. Cada vez que se inicia un Jupyter Notebook se crea un nuevo núcleo.

Cómo usar Jupyter

En la siguiente sesión, aprenderá a usar Jupyter Notebook. Escribirá una simple línea de código para familiarizarse con el entorno de Jupyter.

Paso 1) Añada una carpeta dentro del directorio de trabajo que contendrá todos los blocs de notas que va a crear durante los tutoriales sobre TensorFlow.

Abra la Terminal y escriba

```
mkdir jupyter_tf  
jupyter notebook
```

Explicación del código

- mkdir jupyter_tf: Crear una carpeta nombres jupyter_tf
- cuaderno jupyter: Abrir la aplicación web de Jupyter

A terminal session showing two commands being run:

```
Last login: Tue Apr  3 10:21:30 on ttys002  
[Thomass-MacBook-Pro:~ Thomas$ mkdir jupyter_tf  
[Thomass-MacBook-Pro:~ Thomas$ jupyter notebook
```

Annotations explain the commands:

- An arrow points from the first command to the text: "Crear una nueva carpeta para alojar el bloc de notas".
- An arrow points from the second command to the text: "Abrir la aplicación web de Jupyter".

Paso 2) Puede ver la nueva carpeta dentro del entorno. Haga clic en la carpeta jupyter_tf.

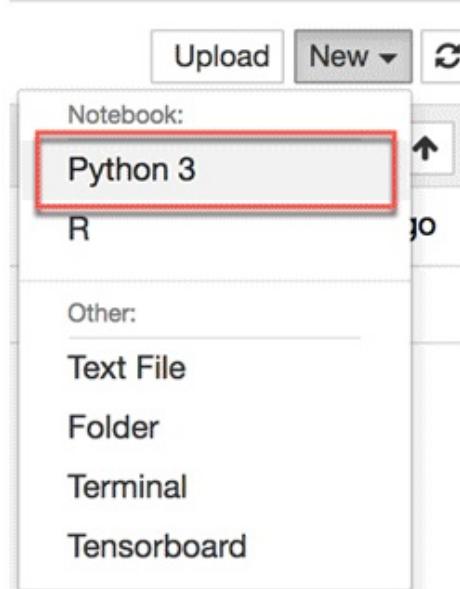
Files Running Clusters

Select items to perform actions on them.

Upload New

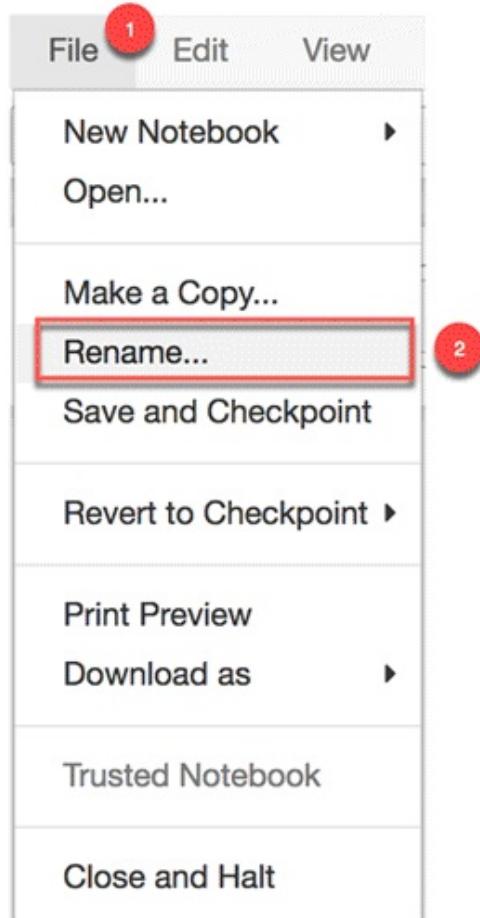
	Name	Last Modified
<input type="checkbox"/>	anaconda3	20 hours ago
<input type="checkbox"/>	Applications	2 months ago
<input type="checkbox"/>	Desktop	2 minutes ago
<input type="checkbox"/>	Docker	2 months ago
<input type="checkbox"/>	Documents	8 days ago
<input type="checkbox"/>	Downloads	2 hours ago
<input type="checkbox"/>	Dropbox	a day ago
<input type="checkbox"/>	Jupyter_notebook	4 minutes ago
<input type="checkbox"/>	jupyter_tf	seconds ago
<input type="checkbox"/>	Movies	4 months ago
<input type="checkbox"/>	Music	4 months ago
<input type="checkbox"/>	Pictures	2 months ago
<input type="checkbox"/>	Public	8 months ago

Paso 3) Dentro de esta carpeta, usted creará su primer cuaderno. Haga clic en el botón **Nuevo y Python 3**.



Paso 4) Usted está dentro del entorno Jupyter. Hasta ahora, su cuaderno se llama Untitled.ipynb. Este es el nombre predeterminado dado por Jupyter. Vamos a cambiar el nombre haciendo clic en **Archivo**

y Cambiar el nombre

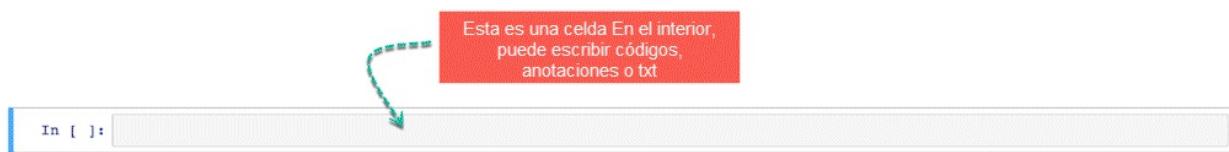


Puede cambiarle el nombre `Introduction_jupyter`



En Jupyter Notebook, escribe códigos, anotaciones o texto dentro de las

celdas.



Dentro de una celda, puede escribir una sola línea de código.

```
In [1]: # Single line
import matplotlib.pyplot as plt
```

o varias líneas Jupyter lee el código una línea tras otra.

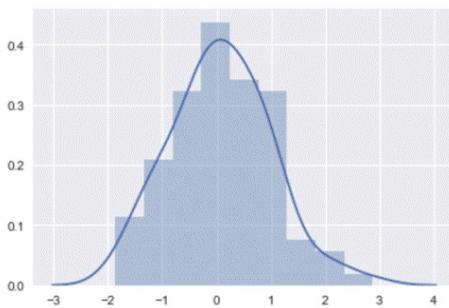
```
In [2]: # Multiple line
import numpy as np
import pandas as pd
from scipy import stats, integrate
```

Por ejemplo, si escribe el siguiente código dentro de una celda.

```
In [6]: # Run the code
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "distributions")))
x = np.random.normal(size=100)
sns.distplot(x)
plt
```

Producirá esta salida.

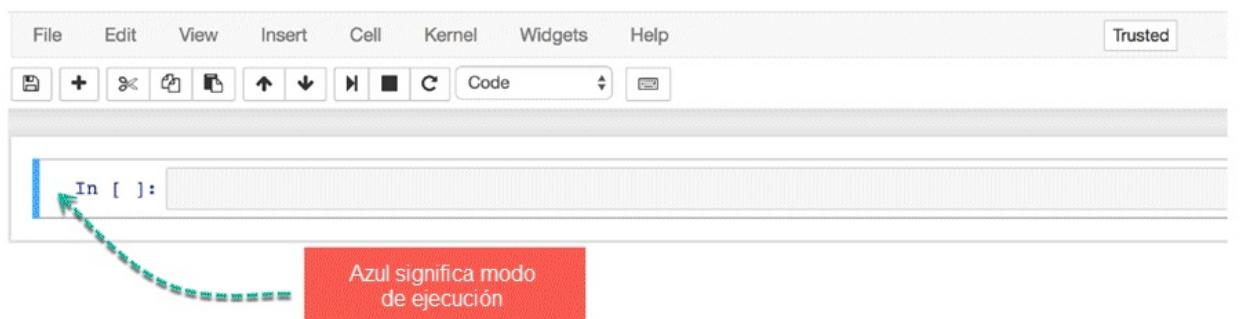
```
Out[6]: <module 'matplotlib.pyplot' from '/Users/Thomas/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py'>
```



Paso 5) Usted está listo para escribir su primera línea de código. Puede notar que la celda tiene dos colores. El color verde significa que estás en el **modo de edición**.



El color azul, sin embargo, indica que está en **modo de ejecución**.



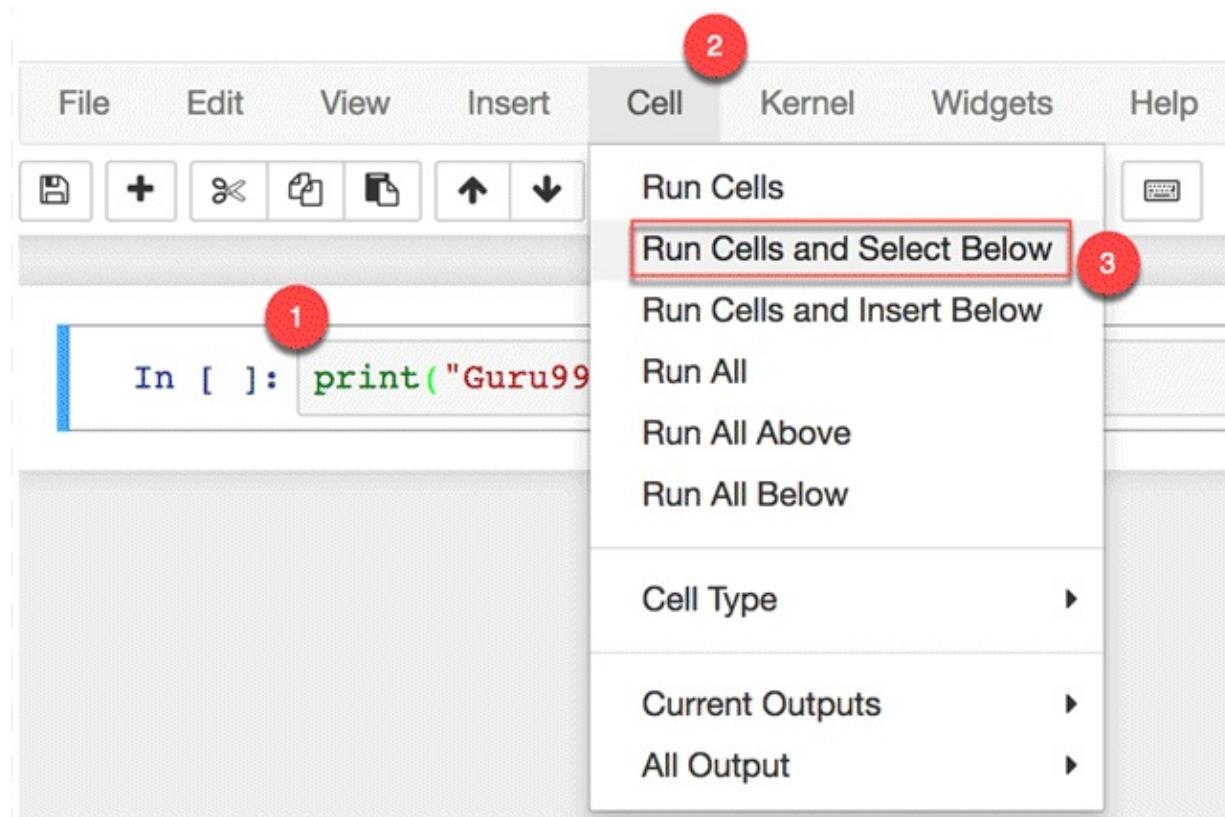
La primera línea de código será imprimir Guru99!. Dentro de la celda, puede escribir

```
print("Guru99!")
```

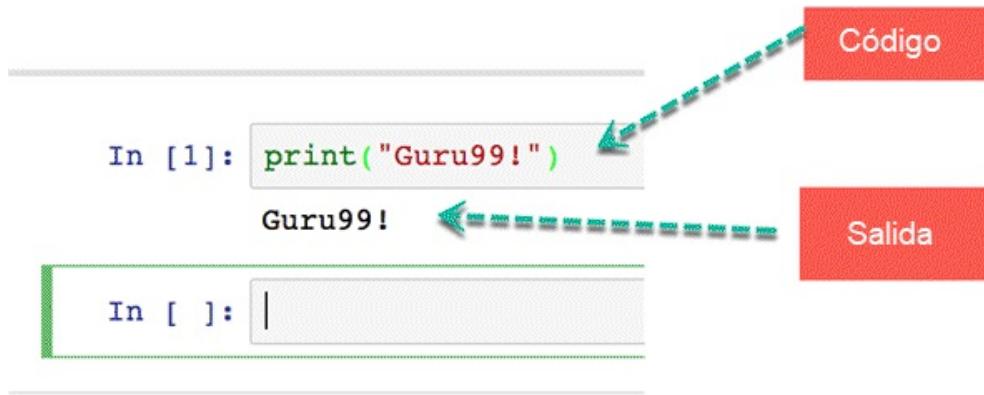
Hay dos formas de ejecutar un código en Jupyter:

- **Haga clic y ejecute**
- **Métodos abreviados de teclado**

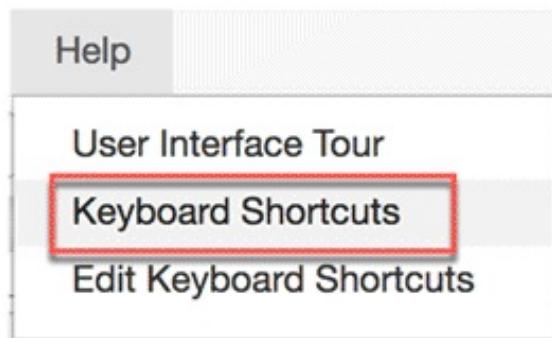
Para ejecutar el código, puede hacer clic en **Celda** y luego **Ejecutar celdas y seleccionar abajo**



Puede ver que el código se imprime debajo de la celda y aparece una nueva celda justo después de la salida.



Una forma más rápida de ejecutar un código es usar el método **Métodos abreviados de teclado**. Para acceder a los métodos abreviados de teclado, vaya a **Ayudar** y **Métodos abreviados de teclado**



Deabajo de la lista de accesos directos de un teclado macOS. Puede editar los accesos directos en el editor.

The screenshot shows the 'Keyboard shortcuts' editor window. At the top, it says 'Keyboard shortcuts' and 'Command Mode (press Esc to enable)'. Below this, a list of keyboard shortcuts is provided:

- F: find and replace
- Esc: enter edit mode
- ⌘ ⌘ F: open the command palette
- ⌘ ⌘ P: open the command palette
- P: open the command palette
- ⇧ ↵: run cell, select below
- ⌘ ↵: run selected cells
- ⌃ ↵: run cell, insert below
- ↑↓: extend selected cells below
- ⇧ J: extend selected cells below
- A: insert cell above
- B: insert cell below
- X: cut selected cells
- C: copy selected cells
- ⇧ V: paste cells above
- V: paste cells below

Some specific keys like 'Esc', 'F', 'P', and the run-cell keys are highlighted with red boxes.

Los siguientes son los accesos directos para Windows

Command Mode (press `Esc` to enable)

`F`: find and replace
`Ctrl-Shift-F`: open the command palette
`Ctrl-Shift-P`: open the command palette
`Enter`: enter edit mode
`P`: open the command palette
`Shift-Enter`: run cell, select below
`Ctrl-Enter`: run selected cells
`Alt-Enter`: run cell and insert below
`Y`: change cell to code
...

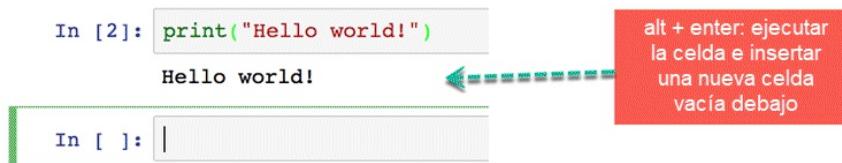
`Shift-Down`: extend selected cells below
`Shift-J`: extend selected cells below
`A`: insert cell above
`B`: insert cell below
`X`: cut selected cells
`C`: copy selected cells
`Shift-V`: paste cells above
`V`: paste cells below
`Z`: undo cell deletion
...

Edit Shortcuts

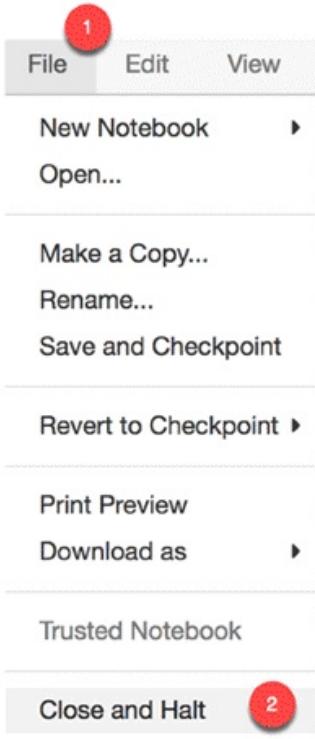
Escribe esta línea

```
print("Hello world!")
```

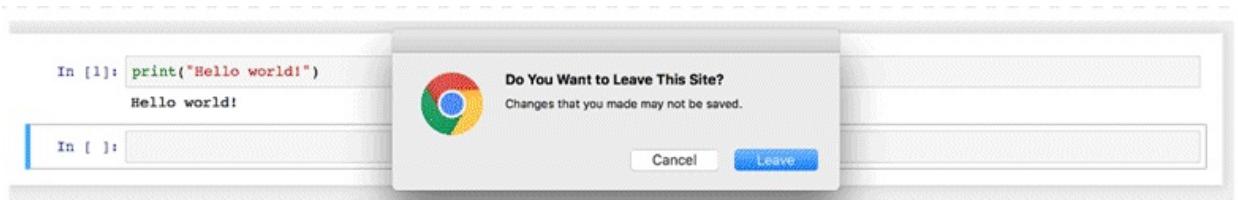
e intente usar los métodos abreviados de teclado para ejecutar el código.
Use alt + enter. ejecutará la celda e insertará una nueva celda vacía a continuación, como lo hizo antes.



Paso 6) Es hora de cerrar el Cuaderno. **Archivo** y haga clic en **Cerrar y detener**



Nota: Jupyter guarda automáticamente el bloc de notas con el punto de control. Si tiene el siguiente mensaje:



Esto significa que Jupyter no guardaba el archivo desde el último punto de control.



In [1]: `print("Hello world!")`

Hello world!

In []:

guardar manualmente el bloc de notas

Se le redirigirá al panel principal. Puede ver que su cuaderno se ha guardado hace un minuto. Puede cerrar sesión de forma segura.



jupyter

Logout

Files Running Clusters

Select items to perform actions on them.

Upload New ↘

Name ↑ Last Modified ↑

seconds ago

a minute ago

/ jupyter_tf

..

Introduction_jupyter.ipynb

El cuaderno se ha guardado hace un minuto

Resumen

- Jupyter notebook es una aplicación web donde puedes ejecutar tus códigos Python y R. Es fácil compartir y ofrecer análisis de datos enriquecidos con Jupyter.
- Para iniciar jupyter, escriba en el terminal: jupyter notebook
- Puedes guardar tu portátil donde quieras
- Una celda contiene su código Python. El núcleo leerá el código uno por uno.
- Puede utilizar el acceso directo para ejecutar una celda. De forma predeterminada: Ctrl + Entrar

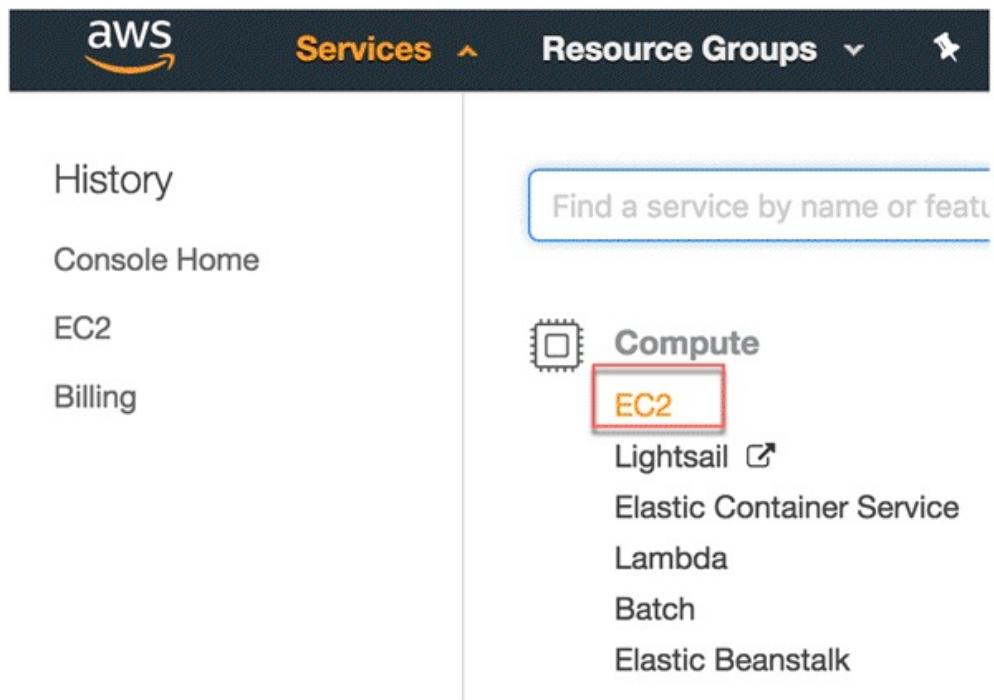
Capítulo 7: Tensorflow en AWS

Este es un tutorial paso a paso, para utilizar Jupyter Notebook en AWS

Si no tiene una cuenta en AWS, cree una cuenta gratuita aquí.

PARTE 1: Configurar un par de claves

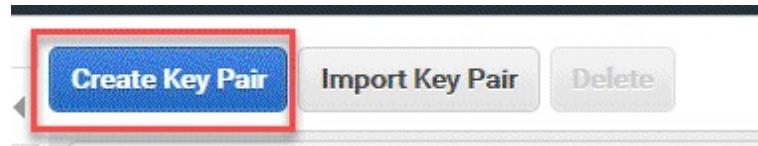
Paso 1) Ir a **Servicios** y encontrar **EC2**



Paso 2) En el panel y haga clic en **Pares de claves**



Paso 3) Haga clic en Crear par de claves



1. Puedes llamarlo clave Docker
2. Haga clic en Crear



Se descarga un nombre de archivo Docker_key.pem.

Key pair name Fingerprint

Docker_key 71:af:12:d0:1e:79:5e:3a:2c:2e:1f:08:2f:71:dc:05:86:10:5e:85

Key Pair: Docker_key

Feedback English (US)

Docker_key.pem

key se crea y se descarga un archivo Docker_key.pem

Paso 4) Copie y péguelo en la clave de carpeta. Lo necesitaremos pronto.

Sólo para usuarios de Mac OS

Este paso solo afecta a usuarios de Mac OS. Para usuarios de Windows o Linux, por favor, diríjase a PARTE 2

Debe establecer un directorio de trabajo que contenga la clave de archivo

En primer lugar, cree una carpeta llamada clave. Para nosotros, se encuentra dentro de la carpeta principal Docker. Luego, establece esta ruta como su directorio de trabajo

```
mkdir Docker/key  
cd Docker/key
```

```
[Thomass-MacBook-Pro:~ Thomas$ mkdir Docker/key  
[Thomass-MacBook-Pro:~ Thomas$ cd Docker/key  
Thomass-MacBook-Pro:key Thomas$ ]
```

Crear nueva carpeta

Hacer que los usuarios/Docker/ clave el nuevo directorio de trabajo

PARTE 2: Configurar un grupo de seguridad

Paso 1) Debe configurar un grupo de seguridad. Puede acceder a él con el panel



Paso 2) Haga clic en Crear grupo de seguridad



Paso 3) En la siguiente pantalla

1. Escriba el nombre del grupo de seguridad “jupyter_docker” y el grupo de seguridad Descripción para Docker
2. Debe agregar 4 reglas encima de
 - ssh: rango de puertos 22, fuente en cualquier lugar
 - http: rango de puertos 80, fuente en cualquier lugar
 - https: rango de puertos 443, fuente en cualquier lugar
 - TCP personalizado: rango de puertos 8888, fuente en cualquier lugar

3. Haga clic en Crear

The screenshot shows the 'Create Security Group' dialog box. Step 1 is highlighted around the top input fields: 'Security group name' (jupyter_docker), 'Description' (Security Group for Docker), and 'VPC' (vpc-620d0207). Step 2 is highlighted around the 'Inbound' rules table, which contains four entries: SSH (TCP 22), HTTP (TCP 80), HTTPS (TCP 443), and a custom rule for port 8888. A green callout points to the port 8888 rule with the text 'Para Jupyter'. Step 3 is highlighted around the 'Create' button at the bottom right.

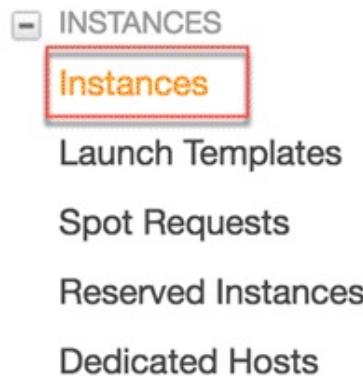
Paso 4) El grupo de seguridad recién creado aparecerá en la lista

The screenshot shows the list of security groups. A new entry is visible: 'sg-c3424a89' (Name), 'jupyter_docker' (Group Name), 'vpc-620d0207' (VPC ID), and 'Security Group for Docker' (Description). This row is highlighted with a red box.

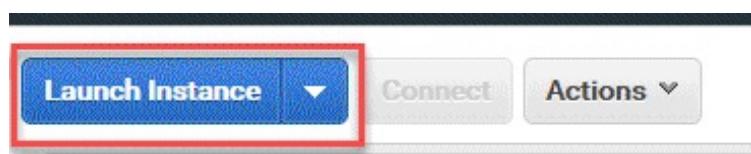
Name	Group ID	Group Name	VPC ID	Description
	sg-c3424a89	jupyter_docker	vpc-620d0207	Security Group for Docker

Parte 3: Instancia de lanzamiento

Finalmente está listo para crear la instancia



Paso 1) Haga clic en Iniciar instancia



El servidor predeterminado es suficiente para sus necesidades. Puede elegir AMI de Amazon Linux. La instancia actual es 2018.03.0.

AMI significa Amazon Machine Image. Contiene la información necesaria para iniciar correctamente una instancia que se ejecuta en un servidor virtual almacenado en la nube.



Tenga en cuenta que AWS tiene un servidor dedicado al aprendizaje profundo, como:

- **AMI de aprendizaje profundo (Ubuntu)**
- **AMI de aprendizaje profundo**
- **Base de aprendizaje profundo AMI (Ubuntu)**

Todos ellos vienen con los últimos binarios de marcos de aprendizaje profundo preinstalados en entornos virtuales separados:

- TensorFlow,
- Caffe
- PyTorch,
- Keras,
- Theano
- CNTK.

Totalmente configurado con nVidia CUDA, CUDNN y NCCL, así como Intel MKL-DNN

Paso 2) Elegir **t2.micro**. Es un servidor de nivel gratuito. AWS ofrece gratuitamente esta máquina virtual equipada con 1 vCPU y 1 GB de memoria. Este servidor proporciona una buena compensación entre el cálculo, la memoria y el rendimiento de la red. Se adapta a bases de datos pequeñas y medianas

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by:	All instance types	Current generation	Show/Hide Columns					
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)								
Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support	
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes	
General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes	
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes	
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes	
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes	
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes	
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes	

Cancel Previous Review and Launch Next: Configure Instance Details 2

Paso 3) Mantenga la configuración predeterminada en la siguiente pantalla y haga clic en Siguiente: Agregar almacenamiento

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Network: vpc-620d0207 (default) | Create new VPC
Subnet: No preference (default subnet in any Availability Zone) | Create new subnet
Auto-assign Public IP: Use subnet setting (Enable)
Placement group: Add instance to placement group.
IAM role: None | Create new IAM role
Shutdown behavior: Stop
Enable termination protection: Protect against accidental termination
Monitoring: Enable CloudWatch detailed monitoring Additional charges apply
Tenancy: Shared - Run a shared hardware instance Additional charges will apply for dedicated tenancy.
T2 Unlimited: Enable Additional charges may apply

Cancel Previous Review and Launch Next: Add Storage

Paso 4) Aumente el almacenamiento a 10 GB y haga clic en Siguiente

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/xvda	snap-07ad5635357af8b3e	10	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	<input type="checkbox"/> Not Encrypted

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Cancel Previous Review and Launch Next: Add Tags

Paso 5) Mantenga la configuración predeterminada y haga clic en Siguiente: Configurar grupo de seguridad

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both.

Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key	(127 characters maximum)	Value	(255 characters maximum)	Instances	Volumes

This resource currently has no tags.
Choose the Add tag button or click to add a Name tag.
Make sure your IAM policy includes permissions to create tags.

Add Tag (Up to 50 tags maximum)

Cancel Previous Review and Launch Next: Configure Security Group

Paso 6) Elija el grupo de seguridad que creó antes, que es

jupyter_docker

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

The screenshot shows the 'Assign a security group' section with two options: 'Create a new security group' (radio button) and 'Select an existing security group' (radio button, highlighted with a red box and number 1). Below is a table of security groups:

Security Group ID	Name	Description	Actions
sg-4ee20c29	default	default VPC security group	Copy to new
sg-c3424a89	jupyter_docker	Security Group for Docker	Copy to new
sg-565dae27	launch-wizard-1	launch-wizard-1 created 2017-06-15T10:09:34.308+05:30	Copy to new

The screenshot shows the 'Inbound rules for sg-c3424a89' section. It lists three rules:

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	::/0	
Custom TCP Rule	TCP	8888	0.0.0.0/0	

At the bottom right, there are buttons: 'Cancel', 'Previous', 'Review and Launch' (highlighted with a red box and number 3).

Paso 7) Revise su configuración y haga clic en el botón de inicio

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

The screenshot shows the 'AMI Details' section of the launch review. It includes:

- AMI:** Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-cfe4b2b0
- Free tier eligible:** The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
- Root Device Type:** ebs
- Virtualization type:** hvm

Below this is a table for 'Security Group ID' with columns: Name and Description. A row for 'jupyter_docker' is shown. At the bottom right are buttons: 'Cancel', 'Previous', and 'Launch' (highlighted with a red box).

Paso 8) El último paso es vincular el par de claves a la instancia.

Select an existing key pair or create a new key pair

X

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Choose an existing key pair

Select a key pair 1

Docker_key ▼

I acknowledge that I have access to the selected private key file (Docker_key.pem), and that 2 without this file, I won't be able to log into my instance.

3 Launch Instances

Paso 8) La instancia se iniciará

 Services ▾ Resource Groups ▾ ★

Launch Status

✓ Your instances are now launching
The following instance launches have been initiated: i-090447e9c051efdce [View launch log](#)

i Get notified of estimated charges
[Create billing alerts](#) to get an email notification when estimated charges on your AWS bill exceed an amount you define (for ex...

How to connect to your instances

Paso 9) A continuación se muestra un resumen de las instancias actualmente en uso.

Reports	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
Limits		i-090447e9c051efdc	t2.micro	us-east-1b	running	Initializing	None	ec2-52-23-241-75.com...	52.23.241.75
Instance: i-090447e9c051efdc Public DNS: ec2-52-23-241-75.compute-1.amazonaws.com									
INSTANCES	Description	Status Checks	Monitoring	Tags					
Instances									
Launch Templates									
Spot Requests									
Reserved Instances									
Dedicated Hosts									
Scheduled Instances									
IMAGES									
AMIs									

Paso 9) Haga clic en Conectar

Launch Instance Connect Actions ▾

Filter by tags and attributes or search by keyword

Name	Instance ID	Instance Type	Availability Zone
	i-090447e9c051efdc	t2.micro	us-east-1b

Encontrará los datos de conexión

Connect To Your Instance

I would like to connect with A standalone SSH client A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (Docker_key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:
`chmod 400 Docker_key.pem`
4. Connect to your instance using its Public DNS:
`ec2-18-188-151-171.us-east-2.compute.amazonaws.com`

Example:

```
ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-171.us-east-2.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Inicie su instancia (usuarios de Mac OS)

Al principio asegúrese de que dentro de la terminal, su directorio de trabajo apunta a la carpeta con el acoplador de archivos de par de claves ejecutar el código

```
chmod 400 docker.pem
```

Abra la conexión con este código.

Hay dos códigos. en algún caso, el primer código evita que Jupyter abra el cuaderno.

En este caso, use el segundo para forzar la conexión.

```
# If able to launch Jupyter
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-
2.compute.amazonaws.com

# If not able to launch Jupyter
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-
2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

La primera vez, se le pedirá que acepte la conexión

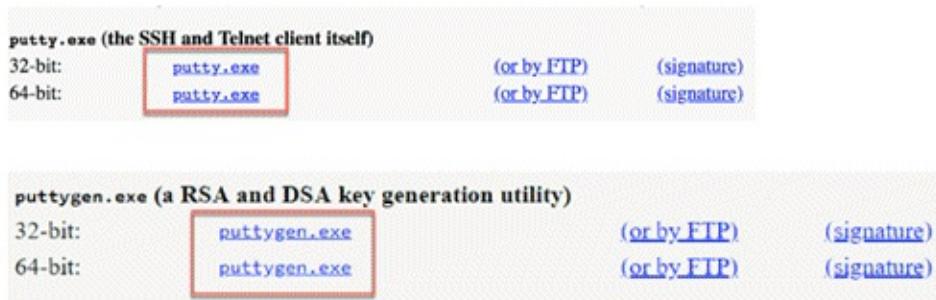
```
thomass-MacBook-Pro:key Thomas$ ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-
171.us-east-2.compute.amazonaws.com
-L 8888:127.0.0.1:8888
The authenticity of host 'ec2-18-188-151-171.us-east-2.compute.amazonaws.com (18
.188.151.171)' can't be established.
ECDSA key fingerprint is SHA256:UuNljpxxnup20pilz0T1LL60Z1o3TdyE86kB6Pmujf0.
Are you sure you want to continue connecting [yes/no]? 
```

Inicie su instancia (usuarios de Windows)

Paso 1) Vaya a este sitio web para descargar PuTTY y PutTyGen PuTTY

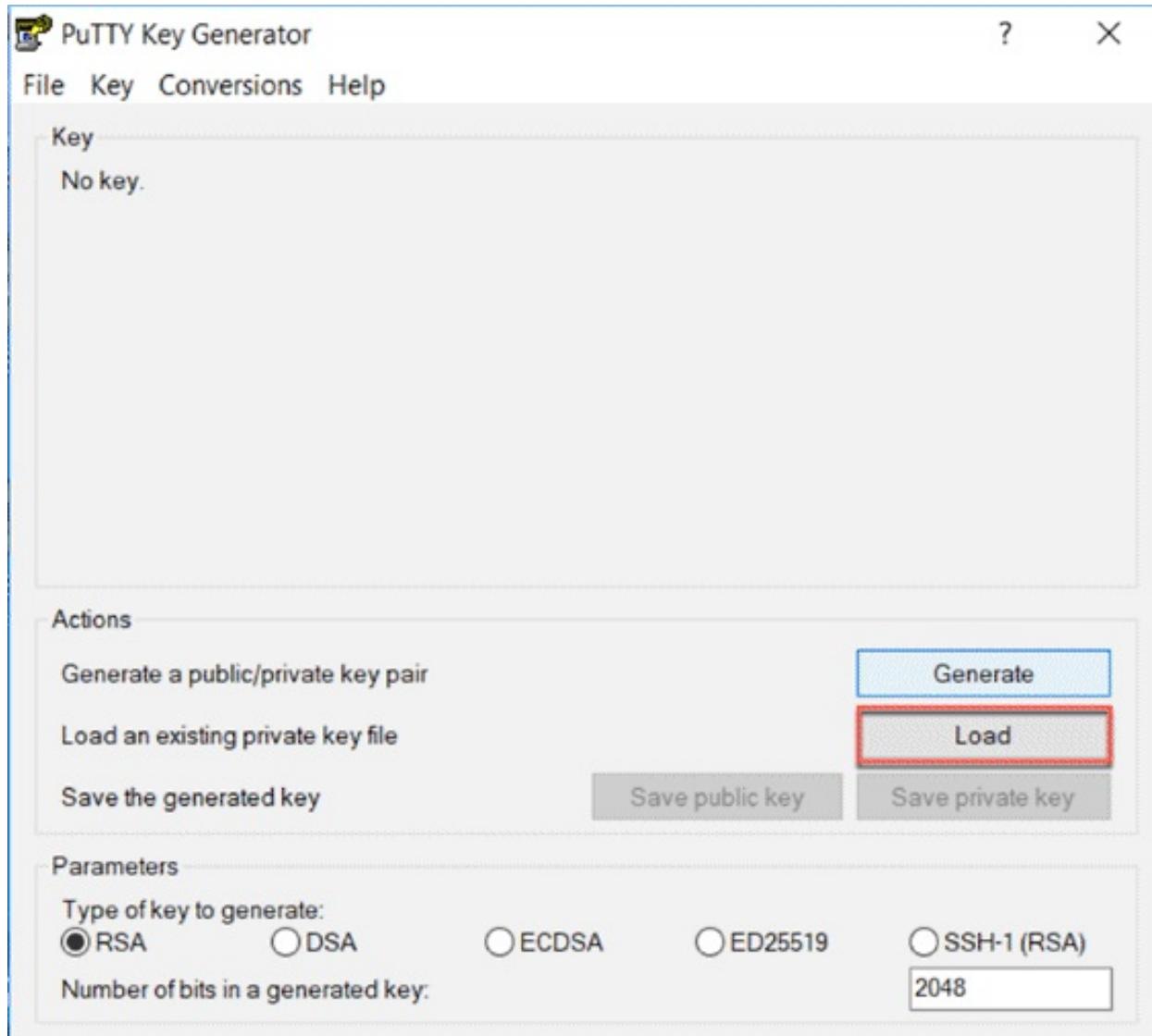
Necesitas descargar

- PuTTY: iniciar la instancia
- putTyGen: convertir el archivo pem a ppk

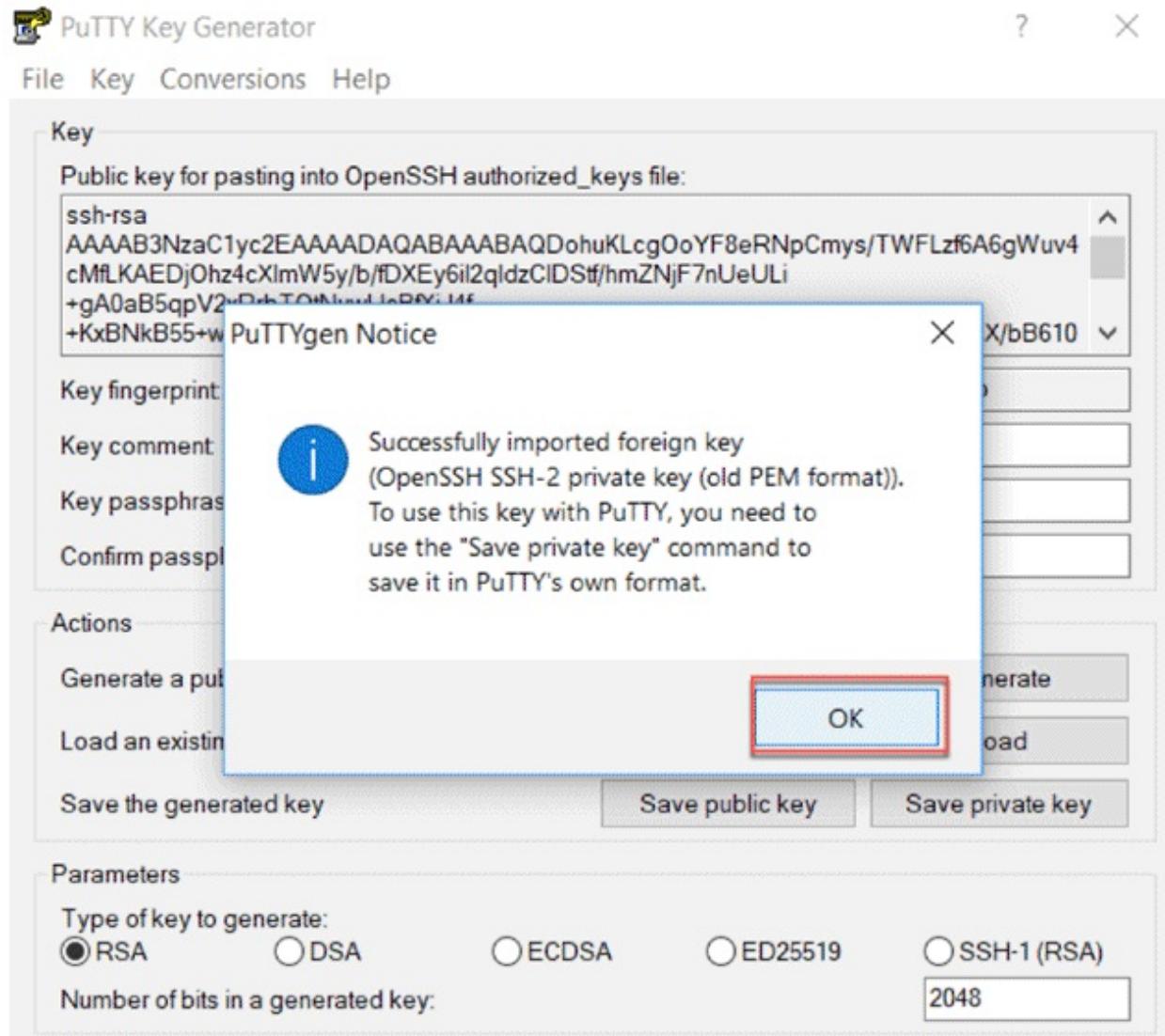


Ahora que ambos software están instalados, debe convertir el archivo.pem a .ppk. PuTTY solo puede leer .ppk. El archivo pem contiene la clave única creada por AWS.

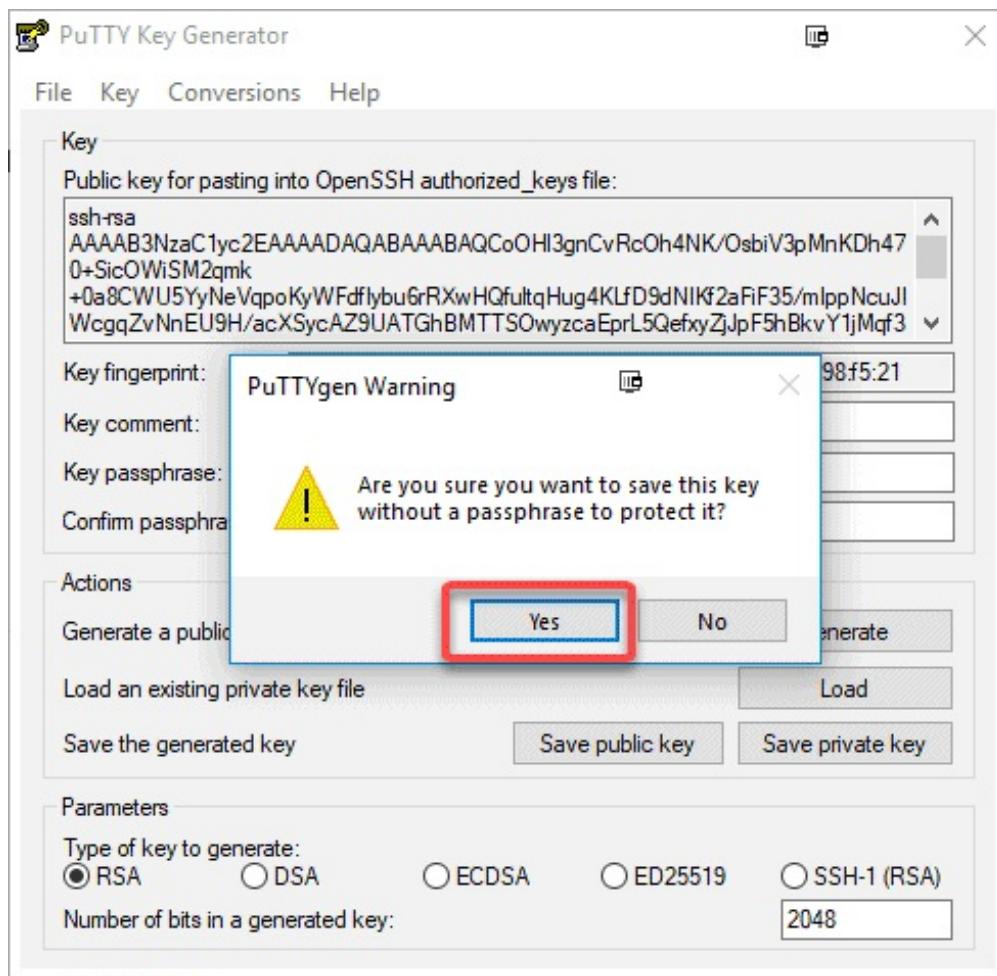
Paso 2) Abra PutTyGen y haga clic en Cargar. Examine la carpeta donde se encuentra el archivo.pem.



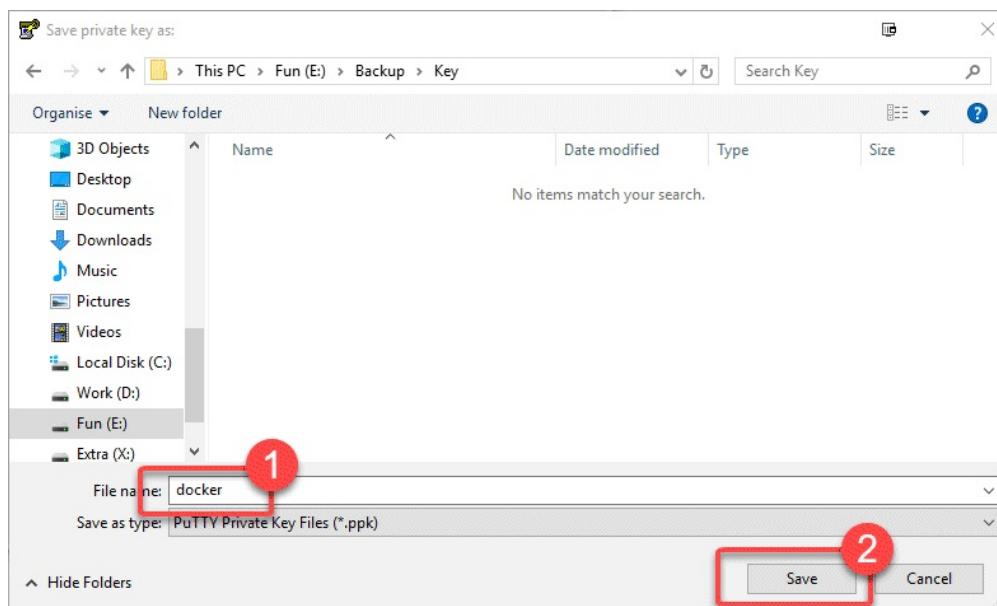
Paso 3) Despu s de cargar el archivo, debe recibir un aviso inform ndole de que la clave se ha importado correctamente. Haga clic en Aceptar



Paso 4) A continuación, haga clic en Guardar clave privada. Se le pregunta si desea guardar esta clave sin una frase de contraseña. Haga clic en sí.



Paso 5) Guarda la clave

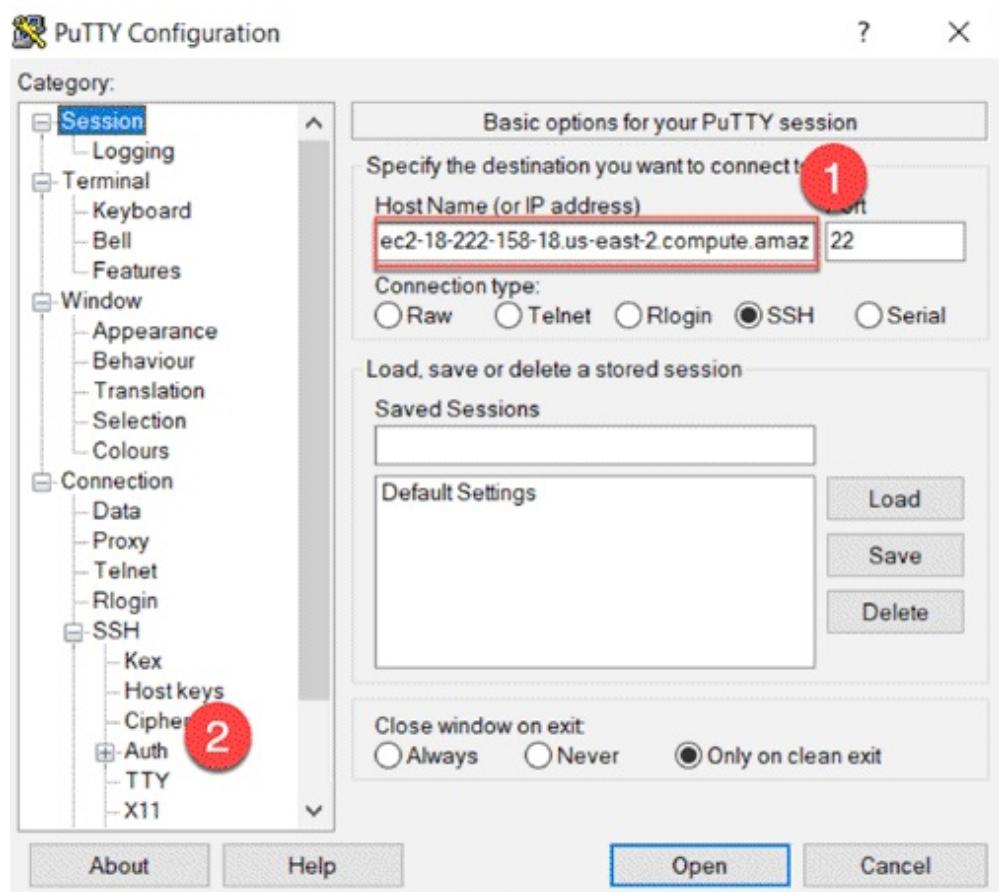


Paso 6) Vaya a AWS y copie el DNS público

4. Connect to your instance using its Public DNS:

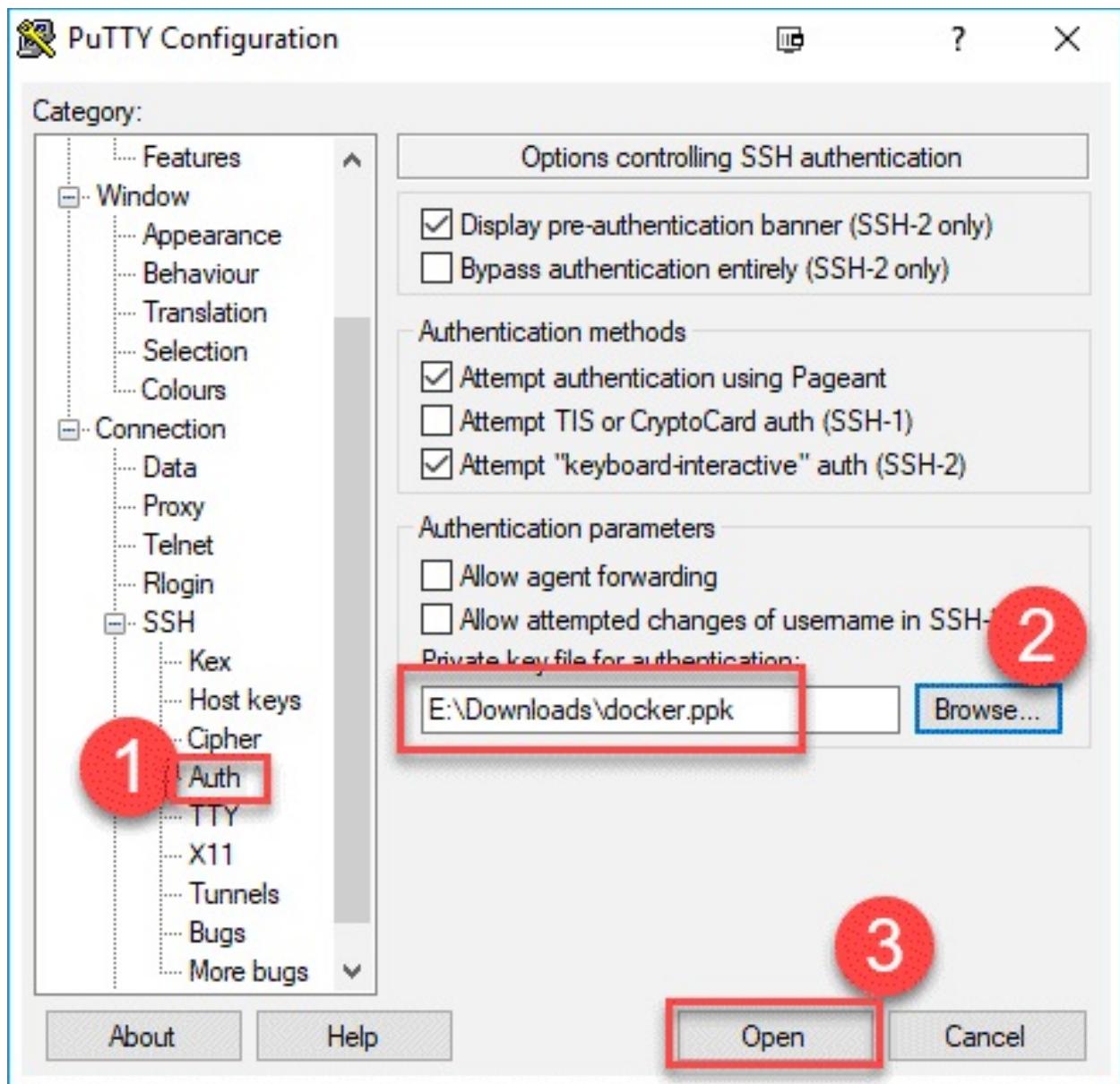
ec2-13-59-162-131.us-east-2.compute.amazonaws.com

Abra PuTTY y pegue el DNS público en el nombre de host



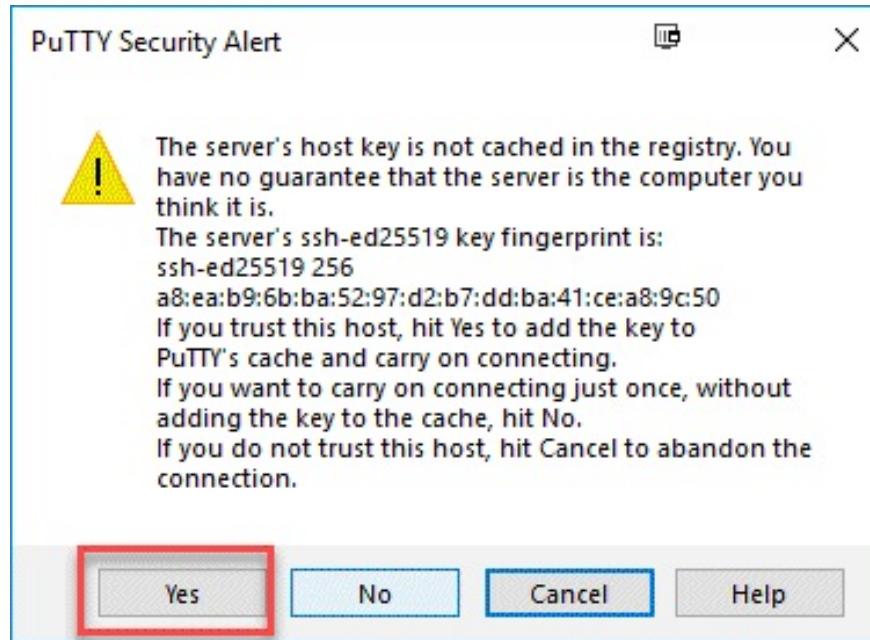
Paso 7)

1. En el panel izquierdo, despliegue SSH y abra Auth
2. Examine la clave privada. Debe seleccionar el archivo.ppk
3. Haga clic en Abrir.



Paso 8)

Cuando termine este paso, se abrirá una nueva ventana. Haga clic en Sí si ve esta ventana emergente



Paso 9)

Necesita iniciar sesión como: ec2-user

A screenshot of a PuTTY terminal window. The title bar says "ec2-18-222-158-18.us-east-2.compute.amazonaws.com - PuTTY". The main pane shows the command "login as: ec2-user" followed by a redacted password field.

Paso 10)

Está conectado a la AMI de Amazon Linux.

A screenshot of a PuTTY terminal window. The title bar says "ec2-user@ip-172-31-57-55:~". The main pane shows the following text:

login as: ec2-user
Authenticating with public key "imported-openssh-key"
____|_____|)
____| _____| / Amazon Linux AMI
<https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/>
2 package(s) needed for security, out of 3 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-57-55 ~]\$

Parte 4: Instalar Docker

Mientras esté conectado con el servidor a través de Putty/Terminal, puede instalar **Docker** contenedor.

Ejecutar los siguientes códigos

```
sudo yum update -y  
sudo yum install -y docker  
sudo service docker start  
sudo user-mod -a -G docker ec2-user  
exit
```

Inicie de nuevo la conexión

```
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-  
2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

Los usuarios de Windows usan SSH como se mencionó anteriormente

Parte 5: Instalar Jupyter

Paso 1) Crear Jupyter con una imagen preconstruida

```
## Tensorflow
docker run -v ~/work:/home/jovyan/work -d -p 8888:8888
jupyter/tensorflow-notebook
## Sparkdocker
run -v ~/work:/home/jovyan/work -d -p 8888:8888 jupyter/pyspark-
notebook
```

Explicación del código

- docker run: Ejecutar la imagen
- v: adjuntar un volumen
- ~/work:/home/jovyan/work: Volume
- 8888:8888: port
- jupyter/datascience -notebook: Imagen

Para otras imágenes precompiladas, vaya aquí

Permitir conservar el portátil Jupyter

```
sudo chown 1000 ~/work
```

Paso 2) Instalar árbol para ver nuestro directorio de trabajo a continuación

```
sudo yum install -y tree
```

```
[ec2-user@ip-172-31-16-239 ~]$ tree
.
└-- work
    1 directory, 0 files
```

Paso 3)

1. Compruebe el contenedor y su nombre (cambia con cada instalación)
Use el comando

```
docker ps
```

2. Obtenga el nombre y use el registro para abrir Jupyter. En el tutorial, el nombre del contenedor es `vigilant_easley`. Usar el comando

```
docker logs vigilant_easley
```

3. Obtener la URL

The screenshot shows a terminal session on an EC2 instance. The user runs `docker ps` (marked with a red circle 1) to list containers. One container is shown, named `vigilant_easley` (marked with a red circle 1). The user then runs `docker logs vigilant_easley` (marked with a red circle 2) to view the Jupyter logs. The logs show the Jupyter server starting and providing a login URL (marked with a red circle 3):

```
[ec2-user@ip-172-31-57-55:~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
90a3c09282d6        jupyter/tensorflow-notebook   "tini -g start-no..."  3 minutes ago    Up 3 minutes      0.0.0.0:8888->8888/tcp
[ec2-user@ip-172-31-57-55:~]$ docker logs vigilant_easley
2
Container must be run with group "root" to update passwd file
Executing the command: jupyter notebook
[I 09:06:47.206 NotebookApp] Writing notebook server cookie secret to /home/joyyan/.local/share/jupyter/runtime/notebook_cookie_secret
[W 09:06:47.671 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[I 09:06:47.724 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab
[I 09:06:47.725 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab
[I 09:06:47.735 NotebookApp] Serving notebooks from local directory: /home/joyyan
[I 09:06:47.735 NotebookApp] The Jupyter Notebook is running at:
[I 09:06:47.735 NotebookApp] http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed
[I 09:06:47.735 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 09:06:47.736 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to log in with a token:
  http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed
[ec2-user@ip-172-31-57-55:~]$
```

Paso 4)

En la URL

`http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed`

Reemplazar `(90a3c09282d6 or 127.0.0.1)` con DNS público de su instancia

Connect To Your Instance

I would like to connect with A standalone SSH client [\(i\)](#) A Java SSH Client directly from my browser (Java required) [\(i\)](#)

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (Docker_key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:
`chmod 400 Docker_key.pem`
4. Connect to your instance using its Public DNS:
`ec2-174-129-135-16.compute-1.amazonaws.com`

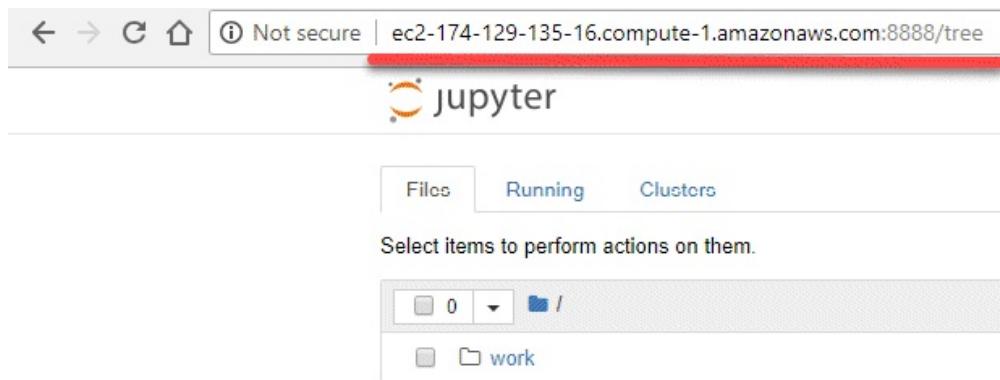
Example:

Paso 5)

La nueva URL se convierte en

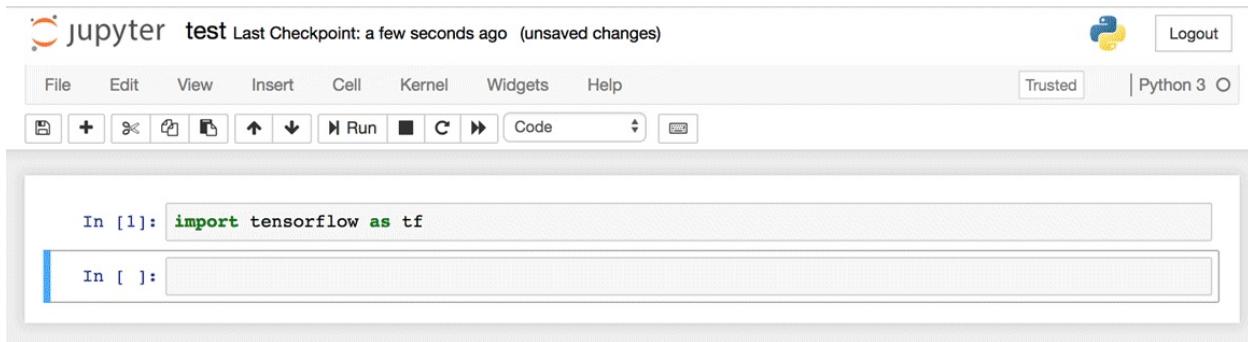
http://ec2-174-129-135-16.compute-1.amazonaws.com:8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed

Paso 6) Copie y pegue la URL en su navegador.



Paso 7)

Puede escribir un nuevo Cuaderno en la carpeta de trabajo

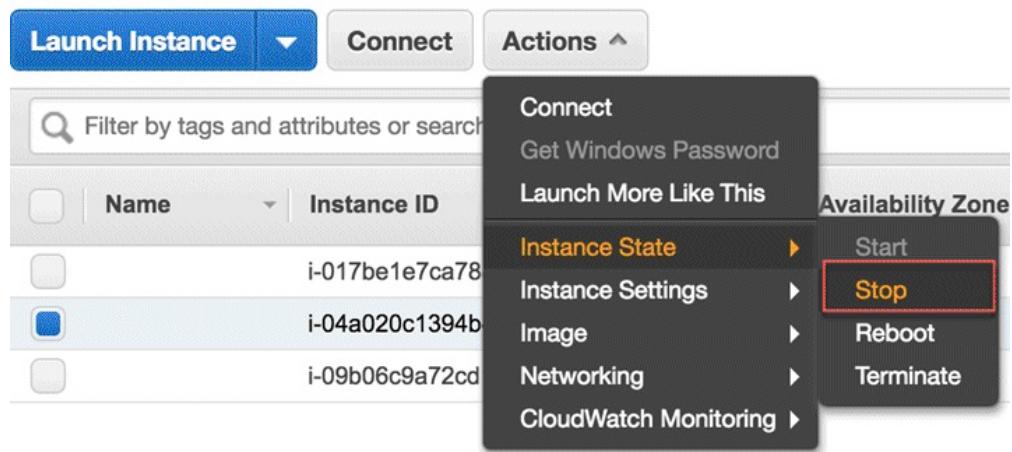


Parte 6: Cerrar conexión

Cerrar la conexión en el terminal

```
exit
```

Vuelva a AWS y detenga el servidor.



Solución de problemas

Si alguna vez el acoplador no funciona, intente reconstruir la imagen usando

```
docker run -v ~/work:/home/jovyan/work -d -p 8888:8888  
jupyter/tensorflow-notebook
```

Capítulo 8: Conceptos básicos de TensorFlow: Tensor, Forma, Tipo, Gráfico, Sesiones y Operadores

¿Qué es un Tensor?

El nombre de Tensorflow se deriva directamente de su marco principal: **Tensor**. En Tensorflow, todos los cálculos involucran tensores **vector** o **matriz** de n dimensiones que representan todos los tipos de datos. Todos los valores de un tensor tienen un tipo de datos idéntico con un conocido (o parcialmente conocido) **forma**. La forma de los datos es la dimensionalidad de la matriz o matriz.

Un tensor se puede originar a partir de los datos de entrada o el resultado de un cálculo. En TensorFlow, todas las operaciones se llevan a cabo dentro de un **gráfico**. El gráfico es un conjunto de cálculos que se lleva a cabo sucesivamente **nodo op** y están conectados entre sí.

El gráfico describe las operaciones y conexiones entre los nodos. Sin embargo, no muestra los valores. El borde de los nodos es el tensor, es decir, una forma de llenar la operación con datos.

En Machine Learning, los modelos se alimentan con una lista de objetos llamada **vectores de entidades**. Un vector de entidad puede ser de cualquier tipo de datos. El vector de entidad suele ser la entrada principal para llenar un tensor. Estos valores fluirán en un nodo op a través del tensor y el resultado de esta operación/cálculo creará un nuevo tensor.

que a su vez será utilizado en una nueva operación. Todas estas operaciones se pueden ver en el gráfico.

Representación de un tensor

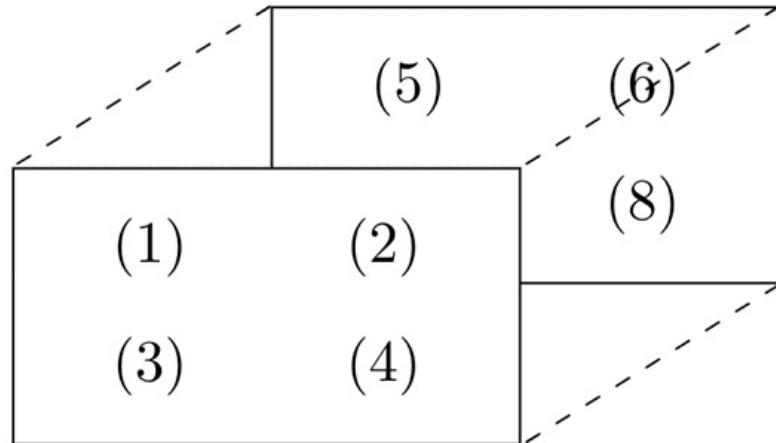
En TensorFlow, un tensor es una colección de vectores de entidades (es decir, matriz) de n-dimensiones. Por ejemplo, si tenemos una matriz 2x3 con valores de 1 a 6, escribimos:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

TensorFlow representa esta matriz como:

```
[[1, 2, 3],  
 [4, 5, 6]]
```

Si creamos una matriz tridimensional con valores de 1 a 8, tenemos:



TensorFlow representa esta matriz como:

```
[ [[1, 2],  
   [[3, 4],  
    [[5, 6],  
     [[7, 8] ]
```

Nota: Un tensor se puede representar con un escalar o puede tener una forma de más de tres dimensiones. Es simplemente más complicado visualizar un nivel de dimensión superior.

Tipos de Tensor

En TensorFlow, todos los cálculos pasan a través de uno o más tensores. Un tensor es un objeto con tres propiedades:

- Una etiqueta única (nombre)
- Una cota (forma)
- Un tipo de datos (dtype)

Cada operación que va a hacer con TensorFlow implica la manipulación de un tensor. Hay cuatro tensores principales que puede crear:

- tf.Variable
- tf.constant
- tf.placeholder
- tf.SparseTensor

En este tutorial, aprenderá a crear una tf.constant y una tf.Variable.

Antes de pasar por el tutorial, asegúrese de activar el entorno conda con TensorFlow. Nós chamamos este ambiente hello-tf.

Para el usuario de macOS:

```
source activate hello-tf
```

Para el usuario de Windows:

```
activate hello-tf
```

Después de haber hecho eso, está listo para importar tensorflow

```
# Import tf
import tensorflow as tf
```

Crear un tensor de dimensión n

Comienza con la creación de un tensor con una dimensión, es decir, un escalar.

Para crear un tensor, puede usar `tf.constant()`

```
tf.constant(value, dtype, name = "")  
arguments  
  
- `value`: Value of n dimension to define the tensor. Optional  
- `dtype`: Define the type of data:  
    - `tf.string`: String variable  
    - `tf.float32`: Flot variable  
    - `tf.int16`: Integer variable  
- "name": Name of the tensor. Optional. By default, `Const_1:0`
```

Para crear un tensor de cota 0, ejecute el siguiente código

```
## rank 0  
# Default name  
r1 = tf.constant(1, tf.int16)  
print(r1)
```

Salida

```
Tensor("Const:0", shape=(), dtype=int16)
```

Tensor("Const:0", shape=(), dtype=int16)

Forma

nombre

Tipo de datos

```
# Named my_scalar
```

```
r2 = tf.constant(1, tf.int16, name = "my_scalar")
print(r2)
```

Salida

```
Tensor("my_scalar:0", shape=(), dtype=int16)
```

Cada tensor se muestra con el nombre del tensor. Cada objeto tensor se define con una etiqueta única (nombre), una dimensión (forma) y un tipo de datos (dtype).

Puede definir un tensor con valores decimales o con una cadena cambiando el tipo de datos.

```
# Decimal
r1_decimal = tf.constant(1.12345, tf.float32)
print(r1_decimal)
# String
r1_string = tf.constant("Guru99", tf.string)
print(r1_string)
```

Salida

```
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("Const_2:0", shape=(), dtype=string)
```

Se puede crear un tensor de dimensión 1 de la siguiente manera:

```
## Rank 1
r1_vector = tf.constant([1,3,5], tf.int16)
print(r1_vector)
r2_boolean = tf.constant([True, True, False], tf.bool)
print(r2_boolean)
```

Salida

```
Tensor("Const_3:0", shape=(3,), dtype=int16)
Tensor("Const_4:0", shape=(3,), dtype=bool)
```

Puede observar que la forma sólo se compone de 1 columna.

Para crear una matriz de 2 dimensiones, debe cerrar los corchetes después de cada fila. Consulte los ejemplos a continuación

```
## Rank 2
r2_matrix = tf.constant([ [1, 2],
                           [3, 4] ], tf.int16)
print(r2_matrix)
```

Salida

```
Tensor("Const_5:0", shape=(2, 2), dtype=int16)
```

La matriz tiene **2** filas y **2** columnas llenas de valores **1, 2, 3, 4**.

Una matriz con **3** dimensiones se construye agregando otro nivel con los corchetes.

```
## Rank 3
r3_matrix = tf.constant([ [[1, 2],
                           [3, 4],
                           [5, 6]] ], tf.int16)
print(r3_matrix)
```

Salida

```
Tensor("Const_6:0", shape=(1, 3, 2), dtype=int16)
```

La matriz se parece a la imagen dos.

Forma del tensor

Al imprimir el tensor, TensorFlow adivina la forma. Sin embargo, puede obtener la forma del tensor con la propiedad shape.

A continuación, se construye una matriz llena con un número de 10 a 15 y se comprueba la forma de m_shape

```
# Shape of tensor
m_shape = tf.constant([
    [10, 11],
    [12, 13],
    [14, 15]
])
m_shape.shape
```

Salida

```
TensorShape([Dimension(3), Dimension(2)])
```

La matriz tiene 3 filas y 2 columnas.

TensorFlow tiene comandos útiles para crear un vector o una matriz llena de 0 o 1. Por ejemplo, si desea crear un tensor 1-D con una forma específica de 10, rellenado con 0, puede ejecutar el siguiente código:

```
# Create a vector of 0
print(tf.zeros(10))
```

Salida

```
Tensor("zeros:0", shape=(10,), dtype=float32)
```

La propiedad también funciona para la matriz. Aquí, se crea una matriz 10x10 llena con 1

```
# Create a vector of 1
print(tf.ones([10, 10]))
```

Salida

```
Tensor("ones:0", shape=(10, 10), dtype=float32)
```

Puede usar la forma de una matriz dada para hacer un vector de unos. La matriz m_shape es una dimensiones 3x2. Puede crear un tensor con 3 filas rellenasadas por uno con el siguiente código:

```
# Create a vector of ones with the same number of rows as m_shape
print(tf.ones(m_shape.shape[0]))
```

Salida

```
Tensor("ones_1:0", shape=(3, ), dtype=float32)
```

Si pasa el valor 1 al corchete, puede construir un vector de unos igual al número de columnas de la matriz m_shape.

```
# Create a vector of ones with the same number of column as m_shape
print(tf.ones(m_shape.shape[1]))
```

Salida

```
Tensor("ones_2:0", shape=(2, ), dtype=float32)
```

Finalmente, puede crear una matriz 3x2 con solo una

```
print(tf.ones(m_shape.shape))
```

Salida

```
Tensor("ones_3:0", shape=(3, 2), dtype=float32)
```

Tipo de datos

La segunda propiedad de un tensor es el tipo de datos. Un tensor solo puede tener un tipo de datos a la vez. Un tensor solo puede tener un tipo de datos. Puede devolver el tipo con la propiedad `dtype`.

```
print(m_shape.dtype)
```

Salida

```
<dtype: 'int32'>
```

En algunas ocasiones, desea cambiar el tipo de datos. En TensorFlow, es posible con el método `tf.cast`.

Ejemplo

A continuación, un tensor flotante se convierte en entero utilizando el método fundido.

```
# Change type of data
type_float = tf.constant(3.123456789, tf.float32)
type_int = tf.cast(type_float, dtype=tf.int32)
print(type_float.dtype)
print(type_int.dtype)
```

Salida

```
<dtype: 'float32'>
<dtype: 'int32'>
```

TensorFlow elige automáticamente el tipo de datos cuando el argumento no se especifica durante la creación del tensor. TensorFlow adivinará cuáles son los tipos de datos más probables. Por ejemplo, si pasa un texto, adivinará que es una cadena y lo convertirá en cadena.

Creando operador

Algunos operadores TensorFlow útiles

Sabes cómo crear un tensor con TensorFlow. Es hora de aprender a realizar operaciones matemáticas.

TensorFlow contiene todas las operaciones básicas. Puedes comenzar con uno simple. Va a utilizar el método TensorFlow para calcular el cuadrado de un número. Esta operación es sencilla porque sólo se requiere un argumento para construir el tensor.

El cuadrado de un número está construido con `tf.sqrt(x)` con `x` como número flotante.

```
x = tf.constant([2.0], dtype = tf.float32)
print(tf.sqrt(x))
```

Salida

```
Tensor("Sqrt:0", shape=(1,), dtype=float32)
```

Nota: La salida devolvió un objeto tensor y no el resultado del cuadrado de 2. En el ejemplo, imprime la definición del tensor y no la evaluación real de la operación. En la siguiente sección, aprenderá cómo funciona TensorFlow para ejecutar las operaciones.

A continuación se muestra una lista de operaciones de uso común. La idea es la misma. Cada operación requiere uno o más argumentos.

- `tf.add(a, b)`
- `tf.subtract(a, b)`
- `tf.multiply(a, b)`

- `tf.div(a, b)`
- `tf.pow(a, b)`
- `tf.exp(a)`
- `tf.sqrt(a)`

Ejemplo

```
# Add
tensor_a = tf.constant([[1,2]], dtype = tf.int32)
tensor_b = tf.constant([[3, 4]], dtype = tf.int32)

tensor_add = tf.add(tensor_a, tensor_b)print(tensor_add)
```

Salida

```
Tensor("Add:0", shape=(1, 2), dtype=int32)
```

Explicación del código

Cree dos tensores:

- un tensor con 1 y 2
- un tensor con 3 y 4

Sumarás ambos tensores.

Notificación: que ambos tensores necesitan tener la misma forma.
Puede ejecutar una multiplicación sobre los dos tensores.

```
# Multiply
tensor_multiply = tf.multiply(tensor_a, tensor_b)
print(tensor_multiply)
```

Salida

```
Tensor("Mul:0", shape=(1, 2), dtype=int32)
```

Variables

Hasta ahora, solo has creado tensores constantes. No es de gran utilidad. Los datos siempre llegan con diferentes valores, para capturar esto, puede usar la clase Variable. Representará un nodo donde los valores siempre cambian.

Para crear una variable, puede utilizar el método `tf.get_variable()`

```
tf.get_variable(name = "", values, dtype, initializer)
argument
- `name` : Name of the variable
- `values` : Dimension of the tensor
- `dtype` : Type of data. Optional
- `initializer` : How to initialize the tensor. Optional
If initializer is specified, there is no need to include the
`values` as the shape of `initializer` is used.
```

Por ejemplo, el siguiente código crea una variable bidimensional con dos valores aleatorios. De forma predeterminada, TensorFlow devuelve un valor aleatorio. Se le asigna un nombre a la variable var

```
# Create a Variable
## Create 2 Randomized values
var = tf.get_variable("var", [1, 2])
print(var.shape)
```

Salida

```
(1, 2)
```

En el segundo ejemplo, se crea una variable con una fila y dos columnas. Debe usar `[1,2]` para crear la dimensión de la variable

Los valores iniciales de este tensor son cero. Por ejemplo, cuando entrena un modelo, necesita tener valores iniciales para calcular el peso de las

entidades. A continuación, establece este valor inicial en cero.

```
var_init_1 = tf.get_variable("var_init_1", [1, 2], dtype=tf.int32,  
initializer=tf.zeros_initializer)  
print(var_init_1.shape)
```

Salida

```
(1, 2)
```

Puede pasar los valores de un tensor constante en una variable. Se crea un tensor constante con el método `tf.constant()`. Utilice este tensor para inicializar la variable.

Los primeros valores de la variable son 10, 20, 30 y 40. El nuevo tensor tendrá una forma de 2x2.

```
# Create a 2x2 matrix  
tensor_const = tf.constant([[10, 20],  
[30, 40]])  
# Initialize the first value of the tensor equals to tensor_const  
var_init_2 = tf.get_variable("var_init_2", dtype=tf.int32,  
initializer=tensor_const)  
print(var_init_2.shape)
```

Salida

```
(2, 2)
```

Marcador de posición

Un marcador de posición tiene el propósito de alimentar el tensor.

Marcador de posición se utiliza para inicializar los datos para fluir dentro de los tensores. Para proporcionar un marcador de posición, debe usar el método `feed_dict`. El marcador de posición se alimentará sólo dentro de una sesión.

En el siguiente ejemplo, verá cómo crear un marcador de posición con el método `tf.placeholder`. En la siguiente sesión, aprenderá a alimentar un marcador de posición con valor real.

La sintaxis es:

```
tf.placeholder(dtype, shape=None, name=None )  
arguments:  
- `dtype`: Type of data  
- `shape`: dimension of the placeholder. Optional. By default,  
shape of the data  
- `name`: Name of the placeholder. Optional  
data_placeholder_a = tf.placeholder(tf.float32, name =  
"data_placeholder_a")  
print(data_placeholder_a)
```

Salida

```
Tensor("data_placeholder_a:0", dtype=float32)
```

Sesión

TensorFlow funciona en torno a 3 componentes principales:

- Gráfico
- Tensor
- Sesión

Componentes	Descripción
Gráfico	El gráfico es fundamental en TensorFlow. Todas las operaciones matemáticas (ops) se realizan dentro de un gráfico. Puede imaginar un gráfico como un proyecto donde se realizan todas las operaciones. Los nodos representan estas operaciones, pueden absorber o crear nuevos tensores.
Tensor	Un tensor representa los datos que progresan entre operaciones. Anteriormente vio cómo inicializar un tensor. La diferencia entre una constante y una variable es que los valores iniciales de una variable cambiarán con el tiempo.
Sesión	Una sesión ejecutará la operación desde el gráfico. Para alimentar el gráfico con los valores de un tensor, debe abrir una sesión. Dentro de una sesión, debe ejecutar un operador para crear una salida.

Los gráficos y las sesiones son independientes. Puede ejecutar una sesión y obtener los valores para usar más adelante para cálculos posteriores.

En el siguiente ejemplo, usted:

- Crear dos tensores
- Crear una operación
- Abrir una sesión
- Imprimir el resultado

Paso 1) Crear dos tensores x e y

```
## Create, run and evaluate a session
x = tf.constant([2])
y = tf.constant([4])
```

Paso 2) Crear el operador multiplicando x e y

```
## Create operator
multiply = tf.multiply(x, y)
```

Paso 3) Abre una sesión. Todos los cálculos se realizarán dentro de la sesión. Cuando haya terminado, debe cerrar la sesión.

```
## Create a session to run the code
sess = tf.Session()
result_1 = sess.run(multiply)
print(result_1)
sess.close()
```

Salida

```
[8]
```

Explicación del código

- `tf.Session()`: Abra una sesión. Todas las operaciones fluirán dentro de las sesiones
- `run(multiply)`: ejecute la operación creada en el paso 2.
- `print(result_1)`: Finalmente, puede imprimir el resultado
- `close()`: Cerrar la sesión

El resultado muestra 8, que es la multiplicación de x e y.

Otra forma de crear una sesión es dentro de un bloque. La ventaja es que cierra automáticamente la sesión.

```
with tf.Session() as sess:  
    result_2 = multiply.eval()  
    print(result_2)
```

Salida

```
[8]
```

En un contexto de la sesión, puede utilizar el método eval () para ejecutar la operación. Es equivalente a run (). Hace que el código sea más legible.

Puede crear una sesión y ver los valores dentro de los tensores que ha creado hasta ahora.

```
## Check the tensors created before  
sess = tf.Session()  
print(sess.run(r1))  
print(sess.run(r2_matrix))  
print(sess.run(r3_matrix))
```

Salida

```
1  
[[1 2]  
 [3 4]]  
[[[1 2]  
 [3 4]  
 [5 6]]]
```

Las variables están vacías por defecto, incluso después de crear un tensor. Debe inicializar la variable si desea usar la variable. El objeto tf.global_variables_initializer () necesita ser llamado para inicializar los valores de una variable. Este objeto inicializará explícitamente todas las variables. Esto es útil antes de entrenar a un modelo.

Puede comprobar los valores de las variables que creó antes. Tenga en cuenta que debe usar run para evaluar el tensor

```
sess.run(tf.global_variables_initializer())
print(sess.run(var))
print(sess.run(var_init_1))
print(sess.run(var_init_2))
```

Salida

```
[[ -0.05356491  0.75867283]]
[[0 0]]
[[10 20]
 [30 40]]
```

Puede usar el marcador de posición que creó antes y alimentarlo con un valor real. Debe pasar los datos al método feed_dict.

Por ejemplo, tomará el poder de 2 del marcador de posición data_placeholder_a.

```
import numpy as np
power_a = tf.pow(data_placeholder_a, 2)
with tf.Session() as sess:
    data = np.random.rand(1, 10)
    print(sess.run(power_a, feed_dict={data_placeholder_a: data})) # Will succeed.
```

Explicación del código

- import numpy as np: Importar biblioteca numpy para crear los datos
- tf.pow (data_placeholder_a, 2): Crear las operaciones
- np.random.rand (1, 10): Crear una matriz aleatoria de datos
- feed_dict = {data_placeholder_a: data}: Alimentar el marcador de posición con datos

Salida

```
[[0.05478134 0.27213147 0.8803037 0.0398424 0.21172127 0.01444725  
0.02584014 0.3763949 0.66022706 0.7565559 ]]
```

Gráfico

TensorFlow depende de un enfoque genial para renderizar la operación. Todos los cálculos se representan con un esquema de flujo de datos. El gráfico de flujo de datos se ha desarrollado para ver las dependencias de datos entre operación individual. La fórmula matemática o el algoritmo se hacen de una serie de operaciones sucesivas. Un gráfico es una forma conveniente de visualizar cómo se coordinan los cálculos.

El gráfico muestra un **nodo** y un **borde**. El nodo es la representación de una operación, es decir, la unidad de cálculo. El borde es el tensor, puede producir un nuevo tensor o consumir los datos de entrada. Depende de las dependencias entre operación individual.

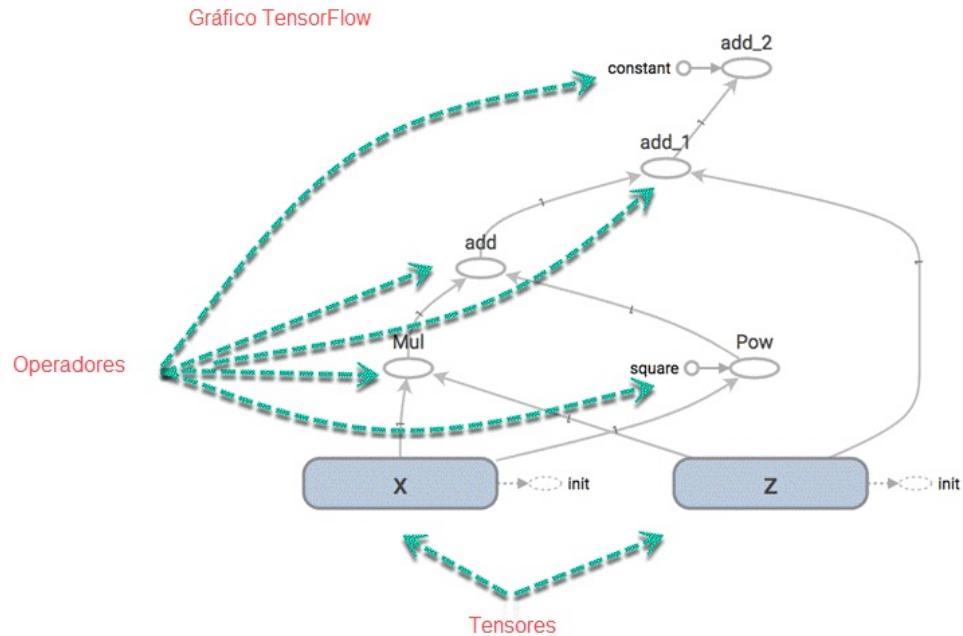
La estructura del gráfico conecta las operaciones (es decir, los nodos) y cómo se alimentan esas operaciones. Tenga en cuenta que el gráfico no muestra la salida de las operaciones, sólo ayuda a visualizar la conexión entre operaciones individuales.

Veamos un ejemplo.

Imagine que desea evaluar la siguiente función:

$$f(x, z) = xz + x^2 + z + 5$$

TensorFlow creará un gráfico para ejecutar la función. El gráfico se ve así:



Se puede ver fácilmente el camino que los tensores tomarán para llegar al destino final.

Por ejemplo, puede ver que la operación add no se puede hacer antes y. El gráfico explica que:

1. calcular y:
2. añadir 1) juntos
3. añadir a 2)
4. añadir 3) a

```
x = tf.get_variable("x", dtype=tf.int32,
initializer=tf.constant([5]))
z = tf.get_variable("z", dtype=tf.int32,
initializer=tf.constant([6]))
c = tf.constant([5], name = "constant")
square = tf.constant([2], name = "square")
f = tf.multiply(x, z) + tf.pow(x, square) + z + c
```

Explicación del código

- x: Inicialice una variable llamada x con un valor constante de 5

- z: Inicialice una variable llamada z con un valor constante de 6
- c: Inicializar un tensor constante llamado c con un valor constante de 5
- cuadrado: Inicializar un tensor constante llamado cuadrado con un valor constante de 2
- f: Construir el operador

En este ejemplo, elegimos mantener los valores de las variables fijos. También creamos un tensor constante llamado c que es el parámetro constante en la función f. Toma un valor fijo de 5. En el gráfico, puede ver este parámetro en el tensor llamado constante.

También construimos un tensor constante para la potencia en el operador tf.pow(). No es necesario. Lo hicimos para que puedas ver el nombre del tensor en el gráfico. Es el círculo llamado cuadrado.

Desde el gráfico, puede entender lo que sucederá de los tensores y cómo puede devolver una salida de 66.

El siguiente código evalúa la función en una sesión.

```
init = tf.global_variables_initializer() # prepare to initialize
all variables
with tf.Session() as sess:
    init.run() # Initialize x and y
    function_result = f.eval()
print(function_result)
```

Salida

[66]

Resumen

TensorFlow funciona en torno a:

- Gráfico: Entorno computacional que contiene las operaciones y los tensores
- Tensores: Representa los datos (o valor) que fluirán en el gráfico. Es el borde del gráfico.
- Sesiones: Permitir la ejecución de las operaciones

Crear un tensor constante

constante	objeto
D0	<code>tf.constant(1, tf.int16)</code>
D1	<code>tf.constant([1,3,5], tf.int16)</code>
D2	<code>tf.constant([[1, 2], [3, 4]],tf.int16)</code>
D3	<code>tf.constant([[[1, 2],[3, 4], [5, 6]]], tf.int16)</code>

Crear un operador

Crear un operador	Objeto
a+b	<code>tf.add(a, b)</code>
a*b	<code>tf.multiply(a, b)</code>

Crear un tensor variable

Crear una variable	objeto
valor aleatorio	<code>tf.get_variable("var", [1, 2])</code>
primer valor inicializado	<code>tf.get_variable("var_init_2", dtype=tf.int32, initializer=[[1, 2], [3, 4]])</code>

Abrir una sesión

Sesión	objeto
Crear una sesión	<code>tf.Session()</code>
Ejecutar una sesión	<code>tf.Session.run()</code>
Evaluar un tensor	<code>variable_name.eval()</code>
Cerrar una sesión	<code>sess.close()</code>
Sesión por bloque	<code>with tf.Session() as sess:</code>

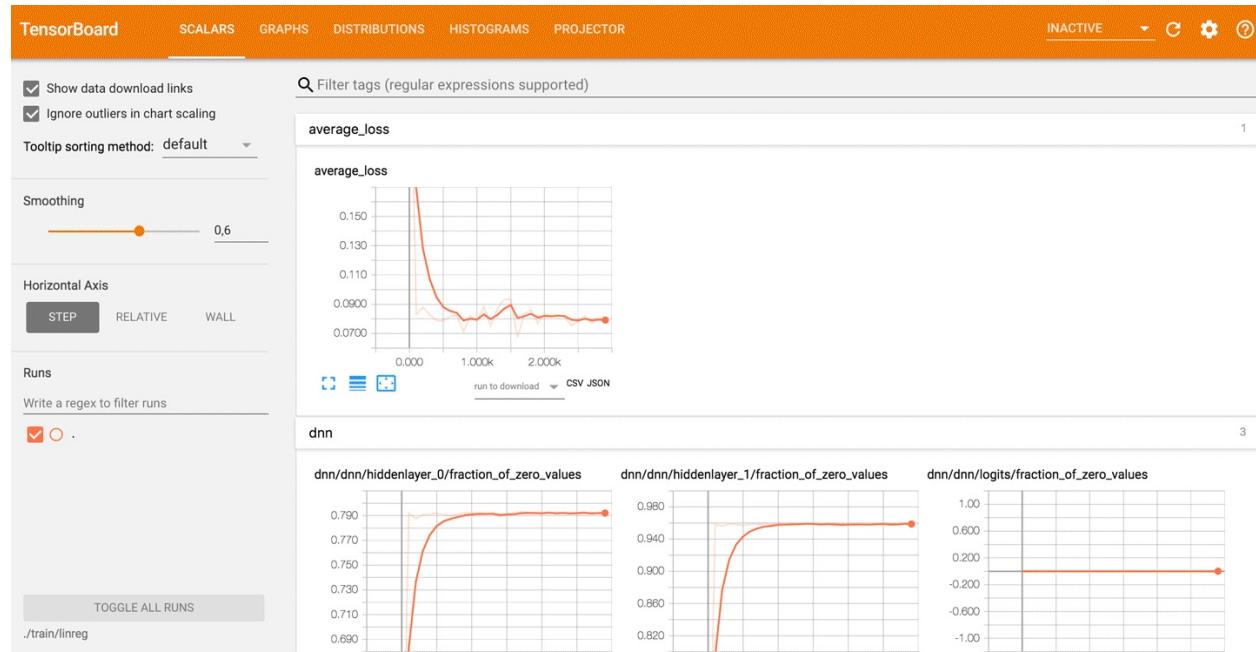
Capítulo 9: Tensorboard: Visualización de gráficos con ejemplo

¿Qué es TensorBoard?

Tensorboard es la interfaz utilizada para visualizar el gráfico y otras herramientas para entender, depurar y optimizar el modelo.

Ejemplo

La imagen de abajo proviene del gráfico que generará en este tutorial. Es el panel principal:



En la imagen de abajo, puede ver el panel de Tensorboard. El panel contiene diferentes fichas, que están vinculadas al nivel de información

que se agrega al ejecutar el modelo.

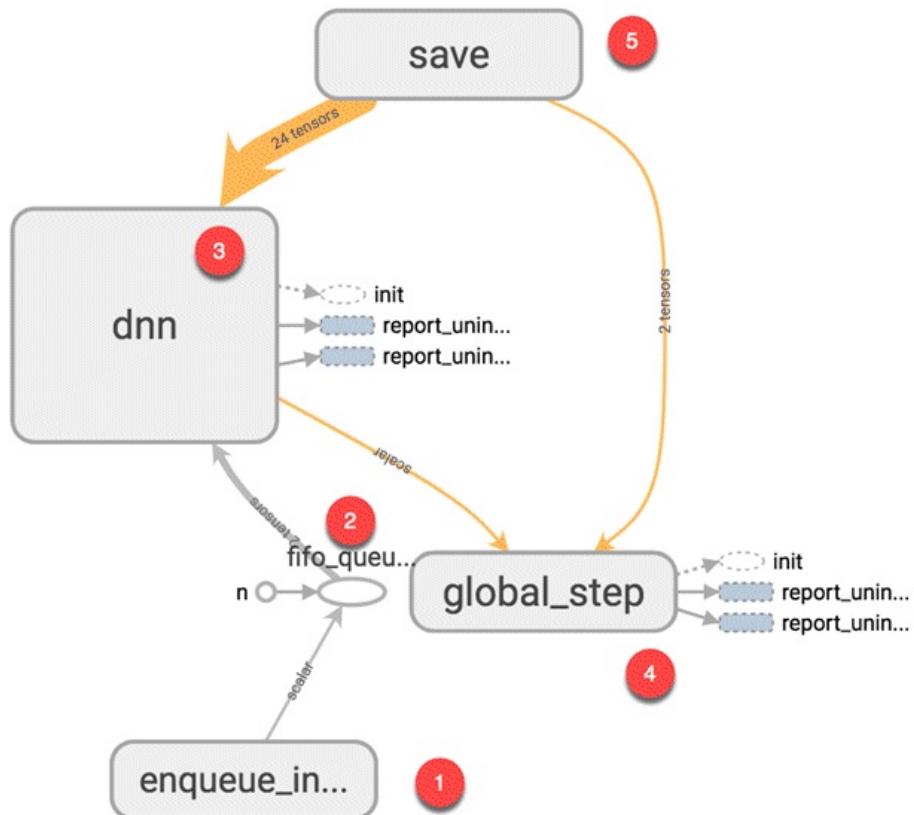


- Escalares: Mostrar diferentes informaciones útiles durante el entrenamiento del modelo
- Gráficos: Mostrar el modelo
- Histograma: Mostrar pesos con un histograma
- Distribución: Muestra la distribución del peso
- Proyector: Muestra análisis de componentes principales y algoritmo T-SNE. La técnica utilizada para la reducción de la dimensionalidad

Durante este tutorial, usted entrenará un modelo simple de aprendizaje profundo. Usted aprenderá cómo funciona en un futuro tutorial.

Si miras el gráfico, puedes entender cómo funciona el modelo.

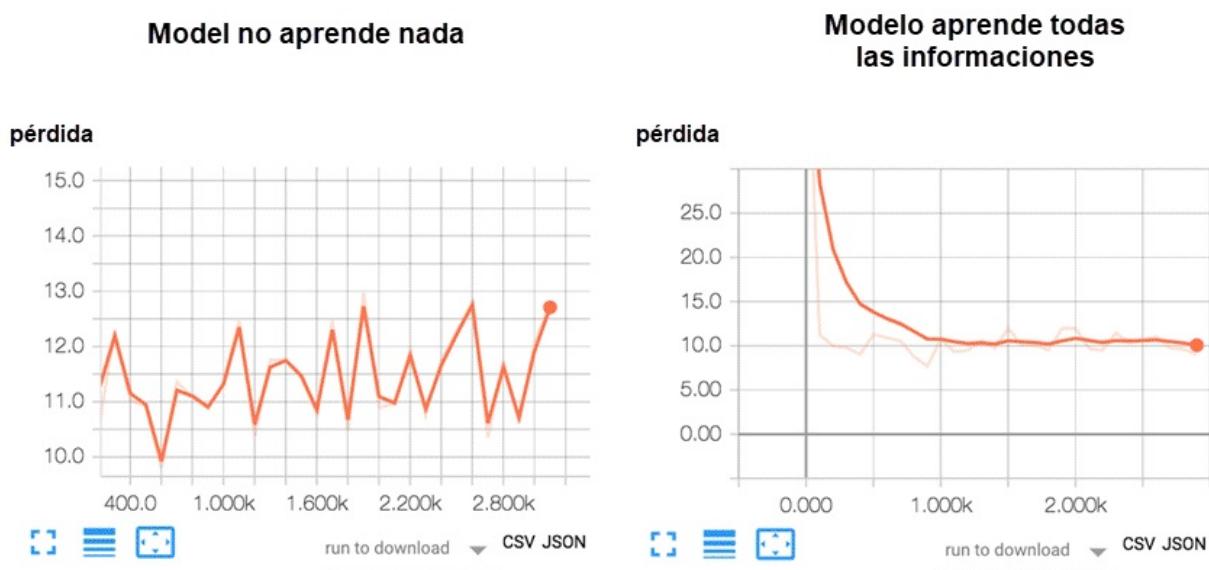
1. Poner en cola los datos en el modelo: Empuje una cantidad de datos igual al tamaño del lote al modelo, es decir, Número de fuente de datos después de cada iteración
2. Alimentar los datos a los tensores
3. Entrenar el modelo
4. Muestra el número de lotes durante el entrenamiento. Guarde el modelo en el disco.



La idea básica detrás del tablero tensorboard es que la red neuronal puede ser algo conocido como una caja negra y necesitamos una herramienta para inspeccionar lo que hay dentro de esta caja. Puedes imaginar el tensorboard como una linterna para empezar a bucear en la red neuronal.

Ayuda a entender las dependencias entre operaciones, cómo se calculan los pesos, muestra la función de pérdida y mucha otra información útil. Cuando juntas todas estas piezas de información, tienes una gran herramienta para depurar y encontrar cómo mejorar el modelo.

Para darle una idea de lo útil que puede ser el gráfico, mire la imagen a continuación:



Una red neuronal decide cómo conectar las diferentes “neuronas” y cuántas capas antes de que el modelo pueda predecir un resultado. Una vez que haya definido la arquitectura, no solo necesita entrenar el modelo, sino también una métrica para calcular la precisión de la predicción. Esta métrica se conoce como **función de pérdida**. El objetivo es minimizar la función de pérdida. En diferentes palabras, significa que el modelo está cometiendo menos errores. Todos los algoritmos de aprendizaje automático repetirán muchas veces los cálculos hasta que la pérdida alcance una línea más plana. Para minimizar esta función de pérdida, debe definir un **tasa de aprendizaje**. Es la velocidad que desea que el modelo aprenda. Si establece una tasa de aprendizaje demasiado alta, el modelo no tiene tiempo para aprender nada. Este es el caso en la imagen de la izquierda. La línea se mueve hacia arriba y hacia abajo, lo que significa que el modelo predice con pura adivinar el resultado. La imagen de la derecha muestra que la pérdida está disminuyendo sobre la iteración hasta que la curva se aplanó, lo que significa que el modelo encontró una solución.

TensorBoard es una gran herramienta para visualizar estas métricas y

resaltar posibles problemas. La red neuronal puede tardar horas o semanas antes de que encuentren una solución. TensorBoard actualiza las métricas con mucha frecuencia. En este caso, no es necesario esperar hasta el final para ver si el modelo entrena correctamente. Puede abrir TensorBoard comprobar cómo va el entrenamiento y hacer el cambio apropiado si es necesario.

En este tutorial, aprenderá a abrir TensorBoard desde el terminal para macOS y la línea de comandos para Windows.

El código se explicará en un futuro tutorial, el foco aquí está en TensorBoard.

En primer lugar, debe importar las bibliotecas que utilizará durante la formación

```
## Import the library
import tensorflow as tf
import numpy as np
```

Se crean los datos Es una matriz de 10000 filas y 5 columnas

```
X_train = (np.random.sample((10000,5)))
y_train = (np.random.sample((10000,1)))
X_train.shape
```

Salida

```
(10000, 5)
```

Los siguientes códigos transforman los datos y crean el modelo.

Tenga en cuenta que la tasa de aprendizaje es igual a 0,1. Si cambia esta tasa a un valor más alto, el modelo no encontrará una solución. Esto es lo que sucedió en el lado izquierdo de la imagen de arriba.

Durante la mayoría de los tutoriales de TensorFlow, utilizará el estimador

TensorFlow. Esta es la API de TensorFlow que contiene todos los cálculos matemáticos.

Para crear los archivos de registro, debe especificar la ruta de acceso. Esto se hace con el argumento `model_dir`.

En el ejemplo siguiente, almacena el modelo dentro del directorio de trabajo, es decir, donde almacena el cuaderno o el archivo python. Dentro de esta ruta, TensorFlow creará una carpeta llamada `tren` con un nombre de carpeta hijo `linreg`.

```
feature_columns = [
    tf.feature_column.numeric_column('x',
shape=X_train.shape[1:])]
DNN_reg =
tf.estimator.DNNRegressor(feature_columns=feature_columns,
# Indicate where to store the log file
    model_dir='train/linreg',
    hidden_units=[500, 300],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.1,
        l1_regularization_strength=0.001
    )
)
```

Salida

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train/linreg',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x1818e63828>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

El último paso consiste en entrenar el modelo. Durante el entrenamiento, TensorFlow escribe información en el directorio del modelo.

```
# Train the estimator
train_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train, shuffle=False, num_epochs=None)
DNN_reg.train(train_input, steps=3000)
```

Salida

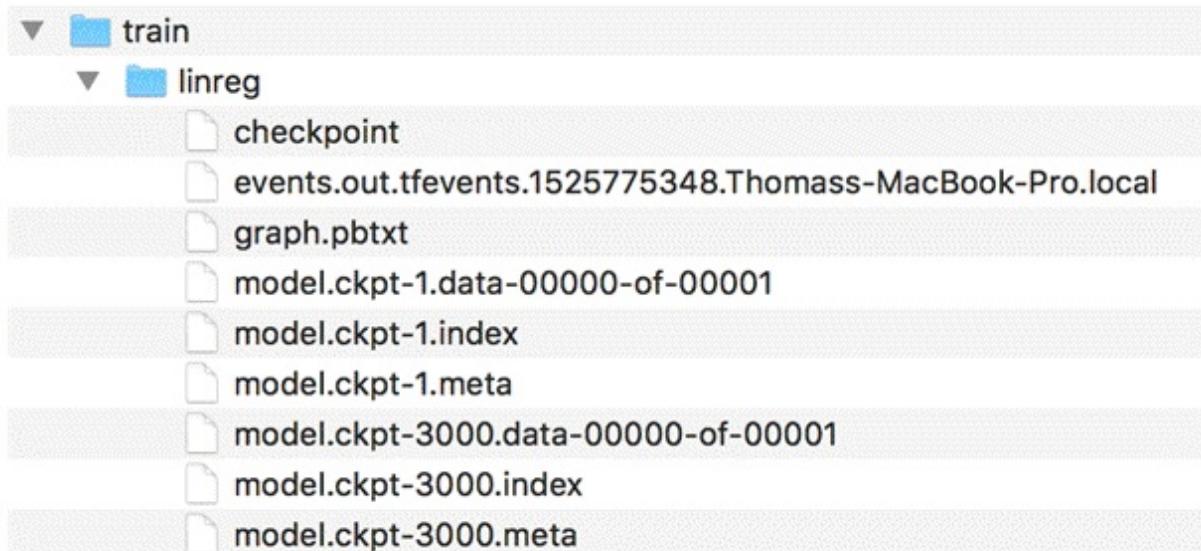
```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
train/linreg/model.ckpt.
INFO:tensorflow:loss = 40.060104, step = 1
INFO:tensorflow:global_step/sec: 197.061
INFO:tensorflow:loss = 10.62989, step = 101 (0.508 sec)
INFO:tensorflow:global_step/sec: 172.487
INFO:tensorflow:loss = 11.255318, step = 201 (0.584 sec)
INFO:tensorflow:global_step/sec: 193.295
INFO:tensorflow:loss = 10.604872, step = 301 (0.513 sec)
INFO:tensorflow:global_step/sec: 175.378
INFO:tensorflow:loss = 10.090343, step = 401 (0.572 sec)
INFO:tensorflow:global_step/sec: 209.737
INFO:tensorflow:loss = 10.057928, step = 501 (0.476 sec)
INFO:tensorflow:global_step/sec: 171.646
INFO:tensorflow:loss = 10.460144, step = 601 (0.583 sec)
INFO:tensorflow:global_step/sec: 192.269
INFO:tensorflow:loss = 10.529617, step = 701 (0.519 sec)
INFO:tensorflow:global_step/sec: 198.264
INFO:tensorflow:loss = 9.100082, step = 801 (0.504 sec)
INFO:tensorflow:global_step/sec: 226.842
INFO:tensorflow:loss = 10.485607, step = 901 (0.441 sec)
INFO:tensorflow:global_step/sec: 152.929
INFO:tensorflow:loss = 10.052481, step = 1001 (0.655 sec)
INFO:tensorflow:global_step/sec: 166.745
INFO:tensorflow:loss = 11.320213, step = 1101 (0.600 sec)
INFO:tensorflow:global_step/sec: 161.854
```

```
INFO:tensorflow:loss = 9.603306, step = 1201 (0.619 sec)
INFO:tensorflow:global_step/sec: 179.074
INFO:tensorflow:loss = 11.110269, step = 1301 (0.556 sec)
INFO:tensorflow:global_step/sec: 202.776
INFO:tensorflow:loss = 11.929443, step = 1401 (0.494 sec)
INFO:tensorflow:global_step/sec: 144.161
INFO:tensorflow:loss = 11.951693, step = 1501 (0.694 sec)
INFO:tensorflow:global_step/sec: 154.144
INFO:tensorflow:loss = 8.620987, step = 1601 (0.649 sec)
INFO:tensorflow:global_step/sec: 151.094
INFO:tensorflow:loss = 10.666125, step = 1701 (0.663 sec)
INFO:tensorflow:global_step/sec: 193.644
INFO:tensorflow:loss = 11.0349865, step = 1801 (0.516 sec)
INFO:tensorflow:global_step/sec: 189.707
INFO:tensorflow:loss = 9.860596, step = 1901 (0.526 sec)
INFO:tensorflow:global_step/sec: 176.423
INFO:tensorflow:loss = 10.695, step = 2001 (0.567 sec)
INFO:tensorflow:global_step/sec: 213.066
INFO:tensorflow:loss = 10.426752, step = 2101 (0.471 sec)
INFO:tensorflow:global_step/sec: 220.975
INFO:tensorflow:loss = 10.594796, step = 2201 (0.452 sec)
INFO:tensorflow:global_step/sec: 219.289
INFO:tensorflow:loss = 10.4212265, step = 2301 (0.456 sec)
INFO:tensorflow:global_step/sec: 215.123
INFO:tensorflow:loss = 9.668612, step = 2401 (0.465 sec)
INFO:tensorflow:global_step/sec: 175.65
INFO:tensorflow:loss = 10.009649, step = 2501 (0.569 sec)
INFO:tensorflow:global_step/sec: 206.962
INFO:tensorflow:loss = 10.477722, step = 2601 (0.483 sec)
INFO:tensorflow:global_step/sec: 229.627
INFO:tensorflow:loss = 9.877638, step = 2701 (0.435 sec)
INFO:tensorflow:global_step/sec: 195.792
INFO:tensorflow:loss = 10.274586, step = 2801 (0.512 sec)
INFO:tensorflow:global_step/sec: 176.803
INFO:tensorflow:loss = 10.061047, step = 2901 (0.566 sec)
INFO:tensorflow:Saving checkpoints for 3000 into
train/linreg/model.ckpt.
INFO:tensorflow:Loss for final step: 10.73032.

<tensorflow.python.estimator.canned.dnn.DNNRegressor at
0x1818e63630>
```

Para el usuario de macOS

Nueva carpeta dentro del directorio de trabajo



Para el usuario de Windows

C:\Users\Admin\train\linreg				
	Name	Date modified	Type	Size
	checkpoint	12-05-2018 12:51 ...	File	
	events.out.tfevents.1526109703.DESKTOP	12-05-2018 12:51 ...	DESKTOP-5OED84...	
	graph.pbtxt	12-05-2018 12:51 ...	PBTXT File	
	model.ckpt-1.data-00000-of-00001	12-05-2018 12:51 ...	DATA-00000-OF-0...	
	model.ckpt-1.index	12-05-2018 12:51 ...	INDEX File	
	model.ckpt-1.meta	12-05-2018 12:51 ...	META File	
	model.ckpt-3000.data-00000-of-00001	12-05-2018 12:51 ...	DATA-00000-OF-0...	
	model.ckpt-3000.index	12-05-2018 12:51 ...	INDEX File	
	model.ckpt-3000.meta	12-05-2018 12:51 ...	META File	

Puede ver esta información en el TensorBoard.

Ahora que tiene los eventos de registro escritos, puede abrir Tensorboard. Tensorboard se ejecuta en el puerto 6006 (Jupyter se ejecuta en el puerto 8888). Puede utilizar el usuario de Terminal para macOS o el indicador Anaconda para el usuario de Windows.

Para el usuario de macOS

```
# Different for you  
cd /Users/Guru99/tuto_TF  
source activate hello-tf!
```

El bloc de notas se almacena en la ruta /Users/Guru99/tuto_TF

Para usuarios de Windows

```
cd C:\Users\Admin\Anaconda3  
activate hello-tf
```

El bloc de notas se almacena en la ruta C:\Users\Admin\Anaconda3

Para lanzar Tensorboard, puede usar este código

Para el usuario de macOS

```
tensorboard --logdir=./train/linreg
```

Para usuarios de Windows

```
tensorboard --logdir=.\train\linreg
```

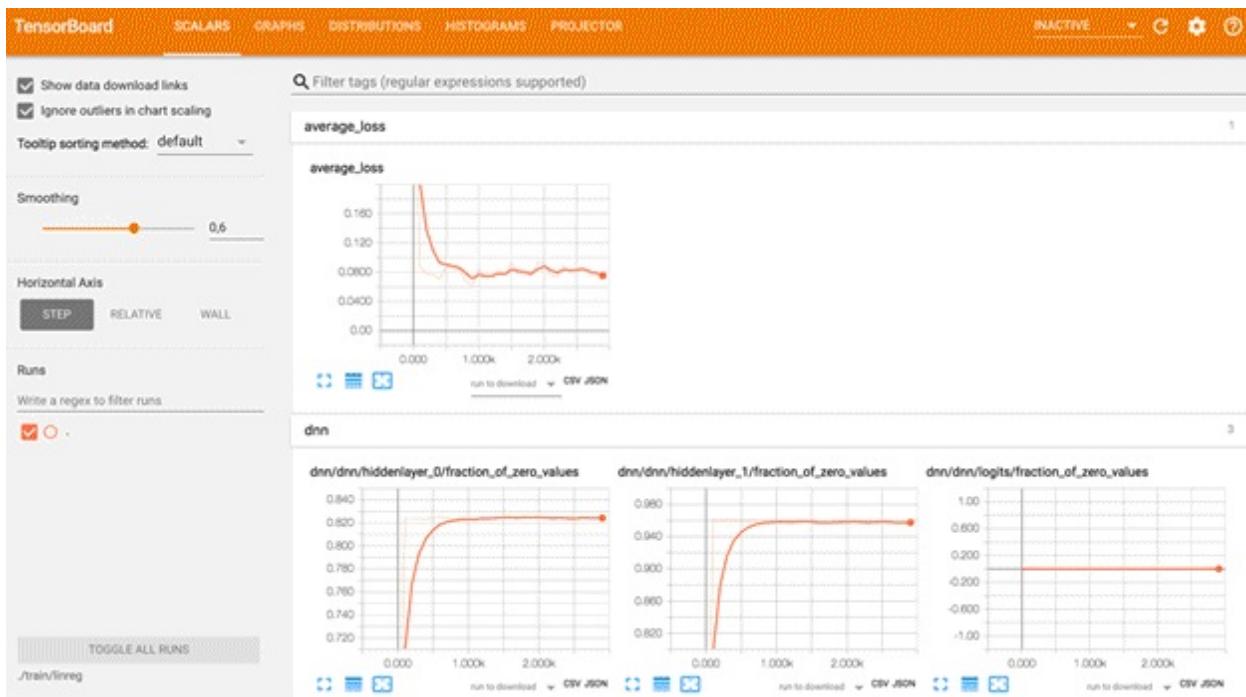
Tensorboard se encuentra en esta URL: <http://localhost:6006>

También podría ubicarse en la siguiente ubicación.

```
(hello-tf) C:\Users\Admin>tensorboard --logdir=.\train\linreg  
2018-05-13 19:08:34.355370: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2  
W0513 19:08:34.371735 Reloader tf_logging.py:121] Found more than one graph event per run, or there was a metagraph containing a graph_def, as well as one or more graph events. Overwriting the graph with the newest event.  
W0513 19:08:34.373239 Reloader tf_logging.py:121] Found more than one metagraph event per run. Overwriting the metagraph with the newest event.  
TensorBoard 1.8.0 at http://DESKTOP-5OED84V:6006 (Press CTRL+C to quit)
```

Copie y pegue la URL en su navegador favorito. Debería ver esto:

Tenga en cuenta que, vamos a aprender a leer el gráfico en el tutorial dedicado al aprendizaje profundo.



Si ves algo como esto:

No graph definition files were found.

To store a graph, create a `tf.summary.FileWriter` and pass the graph either via the constructor, or by calling its `add_graph()` method. You may want to check out the [graph visualizer tutorial](#).

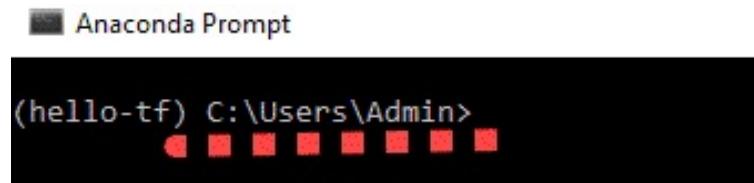
If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the [README](#) and perhaps the [TensorBoard tutorial](#).

If you think TensorBoard is configured properly, please see [the section of the README devoted to missing data problems](#) and consider filing an issue on GitHub.

Significa que Tensorboard no puede encontrar el archivo de registro.
Asegúrese de apuntar el cd a la ruta correcta o compruebe si el evento de registro se ha estado creando. Si no es así, vuelva a ejecutar el código.

Si desea cerrar TensorBoard Presione CTRL + C

Consejo de sombrero: Compruebe su solicitud anaconda para el directorio de trabajo actual,



A screenshot of an Anaconda Prompt window. The title bar says "Anaconda Prompt". The command line shows the path "(hello-tf) C:\Users\Admin>" followed by several red square icons.

El archivo de registro debe crearse en C:\Users\Admin

Resumen:

TensorBoard es una gran herramienta para visualizar tu modelo. Además, se muestran muchas métricas durante el entrenamiento, como la pérdida, la precisión o los pesos.

Para activar Tensorboard, debe establecer la ruta de su archivo:

```
cd /Users/Guru99/tuto_TF
```

Activar el entorno de Tensorflow

```
activate hello-tf
```

Lanzamiento de Tensorboard

```
tensorboard --logdir=.+ PATH
```

Capítulo 10: NumPy

¿Qué es NumPy?

NumPy es una biblioteca de código abierto disponible en Python que ayuda en programación matemática, científica, ingeniería y ciencia de datos. NumPy es una increíble biblioteca para realizar operaciones matemáticas y estadísticas. Funciona perfectamente bien para matrices multidimensionales y multiplicar matrices

Para cualquier proyecto científico, NumPy es la herramienta a conocer. Se ha construido para trabajar con la matriz N-dimensional, álgebra lineal, número aleatorio, transformada de Fourier, etc. Se puede integrar en C/C ++ y Fortran.

NumPy es un lenguaje de programación que se ocupa de matrices y matrices multidimensionales. Además de las matrices y matrices, NumPy admite un gran número de operaciones matemáticas. En esta parte, vamos a revisar las funciones esenciales que usted necesita saber para el tutorial sobre 'TensorFlow'.

¿Por qué usar NumPy?

NumPy es eficiencia de memoria, lo que significa que puede manejar la gran cantidad de datos más accesibles que cualquier otra biblioteca. Además, NumPy es muy conveniente trabajar con, especialmente para la multiplicación y remodelación de matrices. Además de eso, NumPy es rápido. De hecho, TensorFlow y Scikit aprenden a usar la matriz NumPy para calcular la multiplicación de matriz en el back-end.

¿Cómo instalar NumPy?

Para instalar la biblioteca Pandas, consulte nuestro tutorial Cómo instalar TensorFlow. NumPy está instalado de forma predeterminada. En caso remoto, NumPy no instalado-

Puede instalar NumPy usando:

- Anaconda: conda conda install -c anaconda numpy
- En Jupyter Notebook:

```
import sys  
!conda install --yes --prefix {sys.prefix} numpy
```

Importar NumPy y comprobar la versión

El comando para importar numpy es

```
import numpy as np
```

El código anterior cambia el nombre del espacio de nombres Numpy a np. Esto nos permite prefijar la función, los métodos y los atributos Numpy con “np” en lugar de escribir “numpy”. Es el atajo estándar que encontrará en la literatura numpy

Para verificar su versión instalada de Numpy use el comando

```
print (np.__version__)
```

Salida

```
1.14.0
```

Crear una matriz NumPy

La forma más simple de crear una matriz en Numpy es usar Python List

```
myPythonList = [1, 9, 8, 3]
```

Para convertir la lista de Python a una matriz numpy utilizando el objeto np.array.

```
numpy_array_from_list = np.array(myPythonList)
```

Para mostrar el contenido de la lista

```
numpy_array_from_list
```

Salida

```
array([1, 9, 8, 3])
```

En la práctica, no hay necesidad de declarar una lista de Python. La operación se puede combinar.

```
a = np.array([1, 9, 8, 3])
```

NOTA: La documentación de Numpy indica el uso de np.ndarray para crear una matriz. Sin embargo, este el método recomendado

También puede crear una matriz numpy desde una Tupla

Operaciones matemáticas en una matriz

Puede realizar operaciones matemáticas como adiciones, resta, división y multiplicación en una matriz. La sintaxis es el nombre de la matriz seguido de la operación (+ .-, *, /) seguida del operando

Ejemplo:

```
numpy_array_from_list + 10
```

Salida:

```
array([11, 19, 18, 13])
```

Esta operación agrega 10 a cada elemento de la matriz numpy.

Forma de matriz

Puede comprobar la forma de la matriz con la forma de objeto precedida por el nombre de la matriz. De la misma manera, puede verificar el tipo con `dtypes`.

```
import numpy as np
a = np.array([1, 2, 3])
print(a.shape)
print(a.dtype)

(3, )
int64
```

Un entero es un valor sin decimal. Si crea una matriz con decimal, entonces el tipo cambiará a float.

```
#### Different type
b = np.array([1.1, 2.0, 3.2])
print(b.dtype)

float64
```

2 Matriz de dimensiones

Puede agregar una dimensión con un coma ", "

Tenga en cuenta que tiene que estar dentro del corchete []

```
## 2 dimension
c = np.array([(1, 2, 3),
              (4, 5, 6)])
print(c.shape)
(2, 3)
```

3 Matriz de dimensiones

La dimensión más alta se puede construir de la siguiente manera:

```
### 3 dimension
d = np.array([
    [[1, 2, 3],
     [4, 5, 6]],
    [[7, 8, 9],
     [10, 11, 12]]
])
print(d.shape)
(2, 2, 3)
```

np.zeros y np.ones

Puede crear una matriz llena de ceros o unos. Se puede usar cuando inicializó los pesos durante la primera iteración en TensorFlow.

La sintaxis es

```
numpy.zeros(shape, dtype=float, order='C')
```

```
numpy.ones(shape, dtype=float, order='C')
```

Aquí,

Forma: es la forma de la matriz

Dtype: es el tipo de datos. Es opcional. El valor predeterminado es float64

Orden: Por defecto es C, que es un estilo de fila esencial.

Ejemplo:

```
np.zeros((2,2))
```

Salida:

```
array([[0., 0.],  
       [0., 0.]])
```

```
np.zeros((2,2), dtype=np.int16)
```

Salida:

```
array([[0, 0],  
       [0, 0]], dtype=int16)
```

```
## Create 1
```

```
np.ones((1,2,3), dtype=np.int16)
array([[[1, 1, 1],
       [1, 1, 1]]], dtype=int16)
```

Reformar y aplanar datos

En alguna ocasión, debe cambiar la forma de los datos de ancho a largo.

```
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
```

Salida:

```
[[1 2 3]
 [4 5 6]]
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Cuando tratas con alguna red neuronal como convnet, necesitas aplanar la matriz. Puede usar flatten ()

```
e.flatten()
```

```
array([1, 2, 3, 4, 5, 6])
```

hstack y vstack

La biblioteca Numpy también tiene dos funciones convenientes para anexar horizontal o verticalmente los datos. Vamos a estudiarlos con un ejemplo:

```
## Stack
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))

Horizontal Append: [1 2 3 4 5 6]
Vertical Append: [[1 2 3]
 [4 5 6]]
```

Generar números aleatorios

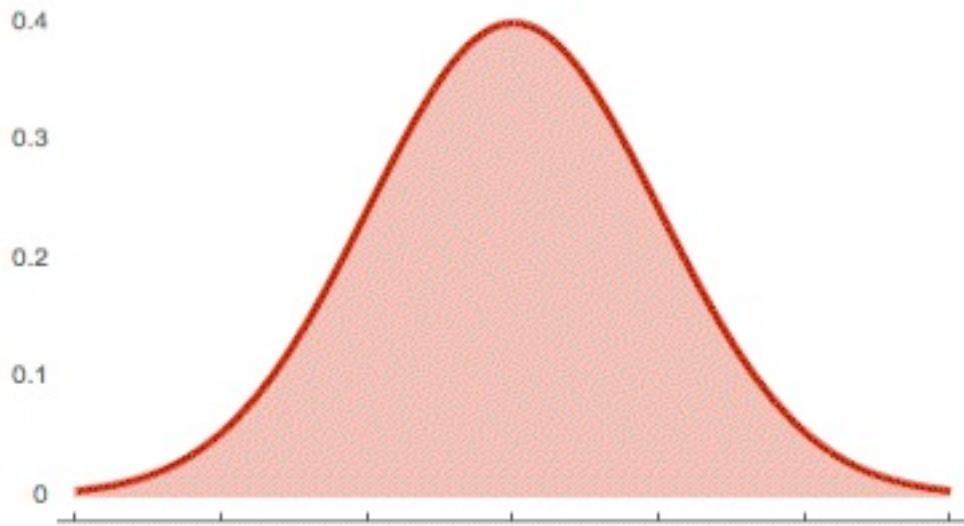
Para generar números aleatorios para la distribución gaussiana, use
numpy .random.normal(loc, scale, size)

Aquí

- Loc: la media. El centro de distribución
- scale: desviación estándar.
- Size: número de devoluciones

```
## Generate random nmber from normal distribution
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
[5.56171852 4.84233558 4.65392767 4.946659  4.85165567 5.61211317
 4.46704244 5.22675736 4.49888936 4.68731125]
```

Si se traza, la distribución será similar a la siguiente parcela



Asarray

Considere la siguiente matriz 2D con cuatro filas y cuatro columnas rellenas por 1

```
A = np.matrix(np.ones((4,4)))
```

Si desea cambiar el valor de la matriz, no puede. La razón es que no es posible cambiar una copia.

```
np.array(A)[2]=2  
print(A)  
[[1. 1. 1. 1.]  
 [1. 1. 1. 1.]  
 [1. 1. 1. 1.]  
 [1. 1. 1. 1.]]
```

Matrix es inmutable. Puede usar asarray si desea agregar modificación en la matriz original. veamos si ocurre algún cambio cuando desea cambiar el valor de las tercera filas con el valor 2

```
np.asarray(A)[2]=2  
print(A)
```

Explicación del código:

np.asarray(A): converts the matrix A to an array

[2]: select the third rows

```
[[1. 1. 1. 1.]  
 [1. 1. 1. 1.]  
 [2. 2. 2. 2.] # new value  
 [1. 1. 1. 1.]]
```

Arreglar

En alguna ocasión, desea crear un valor espaciado uniformemente dentro de un intervalo determinado. Por ejemplo, desea crear valores de 1 a 10; puede usar

Sintaxis:

```
numpy.arange(start, stop, step)
```

- Start: Inicio del intervalo
- Stop: Fin del intervalo
- Step: Espaciado entre valores. El paso predeterminado es 1

Ejemplo:

```
np.arange(1, 11)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Si desea cambiar el paso, puede agregar un tercer número entre paréntesis. Cambiará el paso.

```
np.arange(1, 14, 4)
```

```
array([ 1,  5,  9, 13])
```

Espacio lineal

Linspace proporciona muestras espaciadas uniformemente.

Sintaxis:

```
numpy.linspace(start, stop, num, endpoint)
```

Here,

- Start: Valor inicial de la secuencia
- Stop: Valor final de la secuencia
- Num: Número de muestras que se van a generar. El valor predeterminado es 50
- Endpoint: Si es True (valor predeterminado), stop es el último valor. Si es False, el valor Stop no se incluye.

Por ejemplo, se puede utilizar para crear 10 valores de 1 a 5 espaciados uniformemente.

```
np.linspace(1.0, 5.0, num=10)

array([1.          , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

Si no desea incluir el último dígito en el intervalo, puede establecer el punto final en false

```
np.linspace(1.0, 5.0, num=5, endpoint=False)
```

```
array([1. , 1.8, 2.6, 3.4, 4.2])
```

Espacio de registro

LogSpace devuelve números espaciados incluso en una escala de registro.

Logspace tiene los mismos parámetros que np.linspace.

```
np.logspace(3.0, 4.0, num=4)
array([ 1000. ,  2154.43469003,  4641.58883361, 10000.        ])
```

Finalmente, si desea verificar el tamaño de una matriz, puede usar itemsize

```
x = np.array([1,2,3], dtype=np.complex128)
x.itemsize
```

16

El elemento x tiene 16 bytes.

Indexación y corte

Cortar datos es trivial con numpy. Vamos a cortar el matriz e. Tenga en cuenta que, en Python, debe usar los corchetes para devolver las filas o columnas

```
## Slice
e = np.array([(1,2,3), (4,5,6)])
print(e)
[[1 2 3]
 [4 5 6]]
```

Recuerde con numpy la primera matriz/columna comienza en 0.

```
## First column
print('First row:', e[0])

## Second col
print('Second row:', e[1])
First row: [1 2 3]
Second row: [4 5 6]
```

En Python, como muchos otros idiomas,

- Los valores antes de la coma representan las filas
- El valor de los derechos representa las columnas.
- Si desea seleccionar una columna, debe agregar: antes del índice de columna.
- : significa que desea todas las filas de la columna seleccionada.

```
print('Second column:', e[:,1])
```

```
Second column: [2 5]
```

Para devolver los dos primeros valores de la segunda fila. Utilice: para seleccionar todas las columnas hasta la segunda fila

```
##  
print(e[1, :2])  
[4 5]
```

Función estadística

Numpy está equipado con la robusta función estadística como se indica a continuación

Función	Numpy
Min	np.min()
Max	np.max()
Mean	np.mean()
Median	np.median()
Standard deviation	np.std()

```
## Statistical function
### Min
print(np.min(normal_array))

### Max
print(np.max(normal_array))
### Mean
print(np.mean(normal_array))
### Median
print(np.median(normal_array))
### Sd
print(np.std(normal_array))
```

```
4.467042435266913
5.612113171990201
4.934841002270593
4.846995625786663
0.3875019367395316
```

Producto Dot

Numpy es una poderosa biblioteca para el cálculo de matrices. Por ejemplo, puede calcular el producto de punto con np.dot

```
## Linear algebra
### Dot product: product of two arrays
f = np.array([1,2])
g = np.array([4,5])
### 1*4+2*5
np.dot(f, g)
```

14

Multiplicación Matriz

De la misma manera, puede calcular la multiplicación de matrices con np.matmul

```
### Matmul: matruc product of two arrays
h = [[1,2],[3,4]]
```

```
i = [[5,6],[7,8]]  
### 1*5+2*7 = 19  
np.matmul(h, i)
```

```
array([[19, 22],  
       [43, 50]])
```

Determinante

Por último, pero no menos importante, si necesita calcular el determinante, puede usar np.linalg.det (). Tenga en cuenta que numpy se encarga de la dimensión.

```
## Determinant 2*2 matrix  
### 5*8-7*6np.linalg.det(i)  
  
-2.000000000000005
```

Resumen

A continuación, un resumen de las funciones esenciales utilizadas con NumPy

Objetivo	Código
Crear matriz	array([1,2,3])
imprimir la forma	array(.).shape
remodelar	remodelar
plana una matriz	aplanar
anexar verticalmente	vstack
anexar horizontalmente	hstack
crear una matriz	matriz
crear espacio	arreglar
Crear un espacio lineal	espacio lineal
Crear un espacio de registro	espacio de registro

A continuación se muestra un resumen de la función estadística y aritmética básica

Objetivo	Código
min	min()
max	max()
mean	mean()
median	median()
standard deviation	std()

Aquí está el código completo:

```
import numpy as np

##Create array
### list
myPythonList = [1,9,8,3]
numpy_array_from_list = np.array(myPythonList)
### Directly in numpy
np.array([1,9,8,3])

### Shape
a = np.array([1,2,3])
print(a.shape)

### Type
print(a.dtype)

### 2D array
c = np.array([(1,2,3),
(4,5,6)])
print("2d Array",c)

### 3D array
d = np.array([
[1,2,3],
[4,5,6],
[7,8,9]
])
```

```

[[[1, 2, 3],
[4, 5, 6]],
[[7, 8, 9],
[10, 11, 12]]
])
print("3d Array",d)
### Reshape
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
print("After Reshape",e)
### Flatten
e.flatten()

print("After Flatten",e)

### hstack & vstack
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))

### random number
normal_array = np.random.normal(5, 0.5, 10)
print("Random Number",normal_array)

### asarray
A = np.matrix(np.ones((4,4)))
np.asarray(A)
print("Asrray",A)
### Arrange

print("Arrange",np.arange(1, 11))

### linspace

lin = np.linspace(1.0, 5.0, num=10)
print("Linspace",lin)

### logspace
log1 = np.logspace(3.0, 4.0, num=4)

```

```
print("Logspace",log1)

### Slicing
#### rows
e = np.array([(1,2,3), (4,5,6)])
print(e[0])

#### columns
print(e[:,1])

#### rows and columns
print(e[1, :2])
```

Capítulo 11: Pandas

¿Qué es Pandas?

Pandas es una biblioteca de código abierto que le permite realizar manipulación de datos en Python. La biblioteca Pandas está construida sobre Numpy, lo que significa que Pandas necesita Numpy para operar. Los pandas proporcionan una manera fácil de crear, manipular y discutir los datos. Pandas es también una solución elegante para datos de series temporales.

¿Por qué usar Pandas?

Los científicos de datos utilizan Pandas para sus siguientes ventajas:

- Maneja fácilmente los datos faltantes
- Utiliza **Serie para estructura de datos unidimensional** y **DataFrame para estructura de datos multidimensional**
- Proporciona una forma eficiente de cortar los datos
- Proporciona una forma flexible de fusionar, concatenar o cambiar la forma de los datos
- Incluye una potente herramienta de serie temporal para trabajar con

En pocas palabras, Pandas es una biblioteca útil en el análisis de datos. Se puede utilizar para realizar la manipulación y análisis de datos. Pandas proporcionan estructuras de datos potentes y fáciles de usar, así como los medios para realizar rápidamente operaciones en estas estructuras.

¿Cómo instalar Pandas?

Para instalar la biblioteca Pandas, consulte nuestro tutorial Cómo instalar TensorFlow. Pandas está instalado de forma predeterminada. En caso remoto, pandas no instalado-

Puede instalar Pandas usando:

- Anaconda: conda install -c anaconda pandas
- En Jupyter Notebook:

```
import sys
!conda install --yes --prefix {sys.prefix} pandas
```

¿Qué es un marco de datos?

Un marco de datos es una matriz bidimensional, con ejes etiquetados (filas y columnas). Un marco de datos es una forma estándar de almacenar datos.

El marco de datos es bien conocido por el estadístico y otros profesionales de datos. Un marco de datos es un dato tabular, con filas para almacenar la información y columnas para nombrar la información. Por ejemplo, el precio puede ser el nombre de una columna y 2,3,4 los valores de precio.

Debajo de una imagen de un marco de datos de Pandas:

	Item	Price
0	A	2
1	B	3

¿Qué es una serie?

Una serie es una estructura de datos unidimensional. Puede tener cualquier estructura de datos como integer, float y string. Es útil cuando se desea realizar cálculos o devolver una matriz unidimensional. Una serie, por definición, no puede tener varias columnas. Para este último caso, utilice la estructura del marco de datos.

La serie tiene un parámetro:

- Datos: puede ser una lista, diccionario o valor escalar

```
pd.Series([1., 2., 3.])
```

```
0    1.0
1    2.0
2    3.0
dtype: float64
```

Puede agregar el índice con índice. Ayuda a nombrar las filas. La longitud debe ser igual al tamaño de la columna

```
pd.Series([1., 2., 3.], index=['a', 'b', 'c'])
```

A continuación, se crea una serie de Pandas con un valor que falta para las tercera filas. Tenga en cuenta que los valores que faltan en Python se señalan “NaN”. Puede usar numpy para crear un valor faltante: np.nan artificialmente

```
pd.Series([1, 2, np.nan])
```

Salida

```
0    1.0
1    2.0
2    NaN
```

```
dtype: float64
```

Crear marco de datos

Puede convertir una matriz numpy a un marco de datos pandas con `pd.DataFrame()`. Lo contrario también es posible. Para convertir un marco de datos pandas a una matriz, puede usar `np.array()`

```
## Numpy to pandas
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
print('Data Frame:', df_h)

## Pandas to numpy
df_h_n = np.array(df_h)
print('Numpy array:', df_h_n)
Data Frame:    0   1
0   1   2
1   3   4
Numpy array: [[1 2]
 [3 4]]
```

También puede usar un diccionario para crear un marco de datos Pandas.

```
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}
pd.DataFrame(data=dic)
```

	Edad	Nombre
0	30	Juan
1	40	Herrero

Datos de rango

Los pandas tienen una API conveniente para crear un rango de fechas

```
pd.date_range(date, period, frequency):
```

- El primer parámetro es la fecha de inicio
- El segundo parámetro es el número de períodos (opcional si se especifica la fecha de finalización)
- El último parámetro es la frecuencia: día: 'D', mes: 'M' y año: 'Y'

```
## Create date
# Days
dates_d = pd.date_range('20300101', periods=6, freq='D')
print('Day:', dates_d)
```

Salida

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03',
'2030-01-04', '2030-01-05', '2030-01-06'], dtype='datetime64[ns]',
freq='D')
```

```
# Months
dates_m = pd.date_range('20300101', periods=6, freq='M')
print('Month:', dates_m)
```

Salida

```
Month: DatetimeIndex(['2030-01-31', '2030-02-28', '2030-03-31',
'2030-04-30', '2030-05-31', '2030-06-30'], dtype='datetime64[ns]',
freq='M')
```

Inspección de datos

Puede comprobar la cabeza o la cola del conjunto de datos con head () , o tail () precedida por el nombre del marco de datos del panda

Paso 1) Crear una secuencia aleatoria con numpy. La secuencia tiene 4 columnas y 6 filas

```
random = np.random.randn(6, 4)
```

Paso 2) Luego crea un marco de datos usando pandas.

Utilice dates_m como índice para el marco de datos. Significa que a cada fila se le dará un “nombre” o un índice, correspondiente a una fecha.

Finalmente, se da un nombre a las 4 columnas con las columnas de argumento

```
# Create data with date
df = pd.DataFrame(random,
                   index=dates_m,
                   columns=list('ABCD'))
```

Paso 3) Uso de la función de cabeza

```
df.head(3)
```

	A	B	C	D
2030-01-31	1.139433	1.318510	-0.181334	1.615822
2030-02-28	-0.081995	-0.063582	0.857751	-0.527374
2030-03-31	-0.519179	0.080984	-1.454334	1.314947

Paso 4) Uso de la función de cola

```
df.tail(3)
```

	A	B	C	D
2030-04-30	-0.685448	-0.011736	0.622172	0.104993
2030-05-31	-0.935888	-0.731787	-0.558729	0.768774
2030-06-30	1.096981	0.949180	-0.196901	-0.471556

Paso 5) Una excelente práctica para obtener una pista sobre los datos es usar describe(). Proporciona los recuentos, media, std, min, max y percentil del conjunto de datos.

```
df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000

mean	0.002317	0.256928	-0.151896	0.467601
std	0.908145	0.746939	0.834664	0.908910
min	-0.935888	-0.731787	-1.454334	-0.527374
25%	-0.643880	-0.050621	-0.468272	-0.327419
50%	-0.300587	0.034624	-0.189118	0.436883
75%	0.802237	0.732131	0.421296	1.178404
max	1.139433	1.318510	0.857751	1.615822

Segmentar datos

El último punto de este tutorial es sobre cómo cortar un marco de datos pandas.

Puede utilizar el nombre de la columna para extraer datos de una columna determinada.

```
## Slice
### Using name
df['A']

2030-01-31    -0.168655
2030-02-28     0.689585
2030-03-31     0.767534
2030-04-30     0.557299
2030-05-31    -1.547836
2030-06-30     0.511551
Freq: M, Name: A, dtype: float64
```

Para seleccionar varias columnas, debe usar dos veces el corchete, `[[...,]]`

El primer par de corchetes significa que desea seleccionar columnas, y el segundo par de corchetes indica qué columnas desea devolver.

```
df[['A', 'B']].
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266

2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Puede cortar las filas con:

El siguiente código devuelve las tres primeras filas

```
### using a slice for row
df[0:3]
```

	A	B	C	D
2030-01-31	-0.168655	0.587590	0.572301	-0.031827
2030-02-28	0.689585	0.998266	1.164690	0.475975
2030-03-31	0.767534	-0.940617	0.227255	-0.341532

La función loc se utiliza para seleccionar columnas por nombres. Como de costumbre, los valores antes del coma representan las filas y después se refieren a la columna. Debe utilizar los corchetes para seleccionar más de una columna.

```
## Multi col
df.loc[:, ['A', 'B']]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Hay otro método para seleccionar varias filas y columnas en Pandas. Puede usar iloc []. Este método utiliza el índice en lugar del nombre de las columnas. El siguiente código devuelve el mismo marco de datos que

el anterior

```
df.iloc[:, :2]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Colar una columna

Puede colocar columnas usando pd.drop ()

```
df.drop(columns=['A', 'C'])
```

	B	D
2030-01-31	0.587590	-0.031827
2030-02-28	0.998266	0.475975
2030-03-31	-0.940617	-0.341532
2030-04-30	0.507350	-0.296035
2030-05-31	1.276558	0.523017
2030-06-30	1.572085	-0.594772

Concatenación

Puede concatenar dos DataFrame en Pandas. Puede usar pd.concat()

En primer lugar, debe crear dos DataFrames. Hasta ahora es bueno, ya está familiarizado con la creación de marcos de datos

```
import numpy as np
df1 = pd.DataFrame({'name': ['John', 'Smith', 'Paul'],
                     'Age': ['25', '30', '50']},
                     index=[0, 1, 2])
df2 = pd.DataFrame({'name': ['Adam', 'Smith'],
                     'Age': ['26', '11']},
                     index=[3, 4])
```

Finalmente, concatena los dos DataFrame

```
df_concat = pd.concat([df1, df2])
df_concat
```

	Edad	nombre
0	25	Juan
1	30	Herrero
2	50	Pablo
3	26	Adam
4	11	Herrero

Drop_duplicates

Si un conjunto de datos puede contener duplicados de uso de información, `drop_duplicates` es fácil de excluir filas duplicadas. Puede ver que `df_concat` tiene una observación duplicada, `Smith` aparece dos veces en la columna `name`.

```
df_concat.drop_duplicates('name')
```

	Edad	nombre
0	25	Juan
1	30	Herrero
2	50	Pablo
3	26	Adam

Ordenar valores

Puede ordenar el valor con sort_values

```
df_concat.sort_values('Age')
```

	Edad	nombre
4	11	Herrero
0	25	Juan
3	26	Adam
1	30	Herrero
2	50	Pablo

Cambiar nombre: cambio de índice

Puede usar renombrar para cambiar el nombre de una columna en Pandas. El primer valor es el nombre de la columna actual y el segundo valor es el nuevo nombre de la columna.

```
df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
```

	Edad_ppl	Apellido
0	25	Juan
1	30	Herrero
2	50	Pablo
3	26	Adam
4	11	Herrero

Importar CSV

Durante el tutorial TensorFlow, utilizará el conjunto de datos para adultos. A menudo se utiliza con tarea de clasificación. Está disponible en esta URL <https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data>. Los datos se almacenan en formato CSV. Este conjunto de datos incluye ocho variables categóricas:

Este conjunto de datos incluye ocho variables categóricas:

- clase obrera
- educación
- conyugal
- ocupación
- relación
- raza
- relaciones sexuales
- país_nativo

además, seis variables continuas:

- edad
- Fnlwgt
- núm_de_educación
- ganancia_capital
- capital_pérdida

horas_semana

Para importar un conjunto de datos CSV, puede utilizar el objeto pd.read_csv(). El argumento básico dentro es:

Sintaxis:

```
pandas.read_csv(filepath_or_buffer, sep=',  
' , `names=None` , `index_col=None` , `skipinitialspace=False` )
```

- filepath_or_buffer: ruta o URL con los datos
- sep = ',': Define el delimitador a usar
- `names = None`: Asigne un nombre a las columnas. Si el conjunto de datos tiene diez columnas, debe pasar diez nombres
- `index_col = None`: En caso afirmativo, la primera columna se utiliza como índice de fila
- `skipinitialspace = False`: omita espacios después del delimitador.

Para obtener más información sobre readcsv (), consulte la documentación oficial

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html.

Considere el siguiente ejemplo

```
## Import csv  
import pandas as pd  
## Define path data  
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',  
'education_num', 'marital',  
          'occupation', 'relationship', 'race', 'sex',  
'capital_gain', 'capital_loss',  
          'hours_week', 'native_country', 'label']  
PATH = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"  
df_train = pd.read_csv(PATH,  
                      skipinitialspace=True,  
                      names = COLUMNS,  
                      index_col=False)  
df_train.shape
```

Salida:

```
(32561, 15)
```

Groupby

Una forma fácil de ver los datos es utilizar el método groupby. Este método puede ayudarle a resumir los datos por grupo. A continuación se muestra una lista de métodos disponibles con groupby:

- count: count
- min: min
- max: max
- mean: mean
- median: median
- standard deviation: std
- etc

Dentro de groupby (), puede utilizar la columna que desea aplicar el método.

Echemos un vistazo a una sola agrupación con el conjunto de datos para adultos. Obtendrá la media de todas las variables continuas por tipo de ingresos, es decir, por encima de 50k o por debajo de 50k

```
df_train.groupby(['label']).mean()
```

	age	fnlwgt	education_num	capital_gain	capital_loss
label					
<=50K	36.783738	190340.86517	9.595065	148.752468	53.142921
>50K	44.249841	188005.00000	11.611657	4006.142456	195.001530

Puede obtener el mínimo de edad por tipo de hogar

```
df_train.groupby(['label'])['age'].min()
```

```
label
<=50K    17
>50K    19
Name: age, dtype: int64
```

También puede agrupar por varias columnas. Por ejemplo, puede obtener la ganancia máxima de capital según el tipo de hogar y el estado civil.

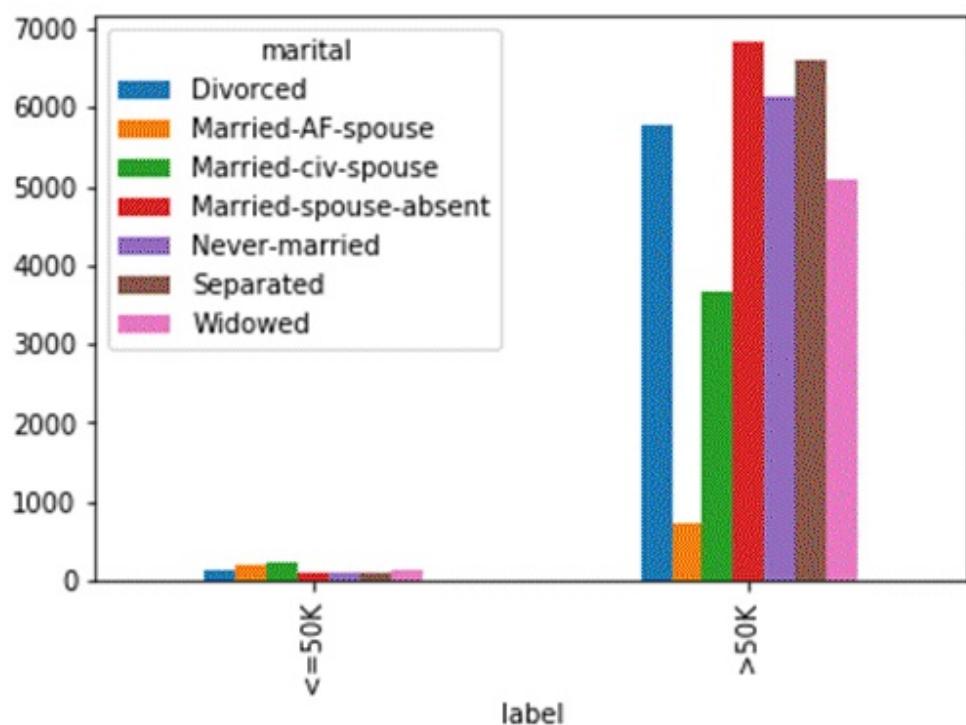
```
df_train.groupby(['label', 'marital'])['capital_gain'].max()
label  marital
<=50K  Divorced          34095
        Married-AF-spouse    2653
        Married-civ-spouse   41310
        Married-spouse-absent 6849
        Never-married        34095
        Separated            7443
        Widowed              6849
>50K   Divorced          99999
        Married-AF-spouse    7298
        Married-civ-spouse   99999
        Married-spouse-absent 99999
        Never-married        99999
        Separated            99999
        Widowed              99999
Name: capital_gain, dtype: int64
```

Puede crear un trazado siguiendo groupby. Una forma de hacerlo es utilizar un trazado después de la agrupación.

Para crear un gráfico más excelente, usará unstack () después de mean () para que tenga el mismo índice multinivel, o unirá los valores por ingresos inferiores a 50k y superiores a 50k. En este caso, la trama tendrá dos grupos en lugar de 14 ($2 * 7$).

Si usa Jupyter Notebook, asegúrese de agregar% matplotlib en línea, de lo contrario, no se mostrará ningún gráfico

```
%matplotlib inline
df_plot = df_train.groupby(['label', 'marital'])
['capital_gain'].mean().unstack()
df_plot
```



Resumen

A continuación se muestra un resumen del método más útil para la ciencia de datos con Pandas

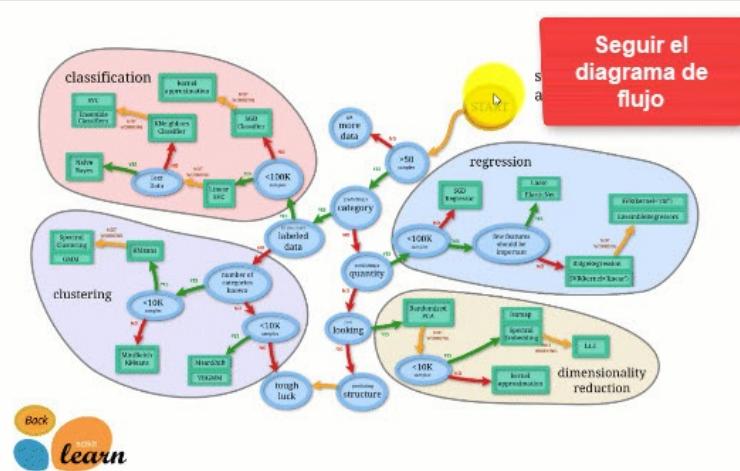
import data	read_csv
create series	Series
Create Dataframe	DataFrame
Create date range	date_range
return head	head
return tail	tail
Describe	describe
slice using name	dataname['columnname']
Slice using rows	data_name[0:5]

Capítulo 12: Scikit-Learn

¿Qué es SciKit-learn?

Scikit-learn es una biblioteca de Python de código abierto para aprendizaje automático. La biblioteca admite algoritmos de última generación como KNN, XGBoost, bosque aleatorio, SVM entre otros. Está construido sobre Numpy. Scikit-learn es ampliamente utilizado en la competencia kaggle, así como en empresas tecnológicas prominentes. SciKit-learn ayuda en el preprocesamiento, reducción de dimensionalidad (selección de parámetros), clasificación, regresión, clustering y selección de modelos.

Scikit-learn tiene la mejor documentación de todas las bibliotecas de código abierto. Le proporciona un gráfico interactivo en http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.



Scikit-learn no es muy difícil de usar y proporciona excelentes resultados. Sin embargo, scikit learn no admite cálculos paralelos. Es posible ejecutar un algoritmo de aprendizaje profundo con él, pero no es una solución

óptima, especialmente si sabe cómo usar TensorFlow.

Descargar e instalar scikit-learn

Opción 1: AWS

scikit-learn se puede utilizar sobre AWS. Consulte La imagen del docker tiene scikit-learn preinstalado.

Para usar la versión del desarrollador, use el comando en Jupyter

```
import sys  
!{sys.executable} -m pip install git+git://github.com/scikit-  
learn/scikit-learn.git
```

Opción 2: Mac o Windows con Anaconda

Para obtener más información sobre la instalación de Anaconda, consulte <https://www.guru99.com/download-install-tensorflow.html>

Recientemente, los desarrolladores de scikit han lanzado una versión de desarrollo que aborda el problema común frente a la versión actual. Nos pareció más conveniente usar la versión del desarrollador en lugar de la versión actual.

Si instaló scikit-learn con el entorno conda, siga el paso para actualizar a la versión 0.20

Paso 1) Activar el entorno tensorflow

```
source activate hello-tf
```

Paso 2) Eliminar scikit lean usando el comando conda

```
conda remove scikit-learn
```

Paso 3) Instale scikit aprender la versión del desarrollador junto con las

bibliotecas necesarias.

```
conda install -c anaconda git  
pip install Cython  
pip install h5py  
pip install git+git://github.com/scikit-learn/scikit-learn.git
```

NOTA: Windows usado necesitará instalar Microsoft Visual C + + 14.
Puede obtenerlo desde aquí

Aprendizaje automático con scikit-learn

Este tutorial se divide en dos partes:

1. Aprendizaje automático con scikit-learn
2. Cómo confiar en tu modelo con LIME

La primera parte detalla cómo construir una tubería, crear un modelo y ajustar los hiperparámetros, mientras que la segunda parte proporciona el estado más avanzado en términos de selección de modelos.

Paso 1) Importar los datos

Durante este tutorial, utilizará el conjunto de datos para adultos. Para conocer los antecedentes de este conjunto de datos, consulte Si está interesado en saber más sobre las estadísticas descriptivas, utilice las herramientas de buceo y visión general. Consulte este tutorial para obtener más información sobre Dive and Overview

Importa el conjunto de datos con Pandas. Tenga en cuenta que debe convertir el tipo de las variables continuas en formato float.

Este conjunto de datos incluye ocho variables categóricas:

Las variables categóricas se enumeran en CATE_Features

- clase obrera
- educación
- conyugal
- ocupación
- relación
- raza
- relaciones sexuales
- país_nativo

además, seis variables continuas:

Las variables continuas se enumeran en CONTI_FEATURES

- edad
- fnlwgt
- núm_de_educación
- ganancia_capital

- capital_pérdida
- horas_semana

Tenga en cuenta que llenamos la lista a mano para que tenga una mejor idea de qué columnas estamos usando. Una forma más rápida de construir una lista de categóricas o continuas es usar:

```
## List Categorical
CATE_FEATURES =
df_train.iloc[:, :-1].select_dtypes('object').columns
print(CATE_FEATURES)

## List continuous
CONTI_FEATURES = df_train._get_numeric_data()
print(CONTI_FEATURES)
```

Aquí está el código para importar los datos:

```
# Import dataset
import pandas as pd

## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital',
    'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss',
    'hours_week', 'native_country', 'label']
### Define continuous list
CONTI_FEATURES = ['age', 'fnlwgt', 'capital_gain', 'education_num',
'capital_loss', 'hours_week']
### Define categorical list
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
'relationship', 'race', 'sex', 'native_country']

## Prepare the data
features = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital',
    'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss',
    'hours_week', 'native_country']

PATH = "https://archive.ics.uci.edu/ml/machine-learning-
```

```

databases/adult/adult.data"

df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
df_train[CONTI_FEATURES]
=df_train[CONTI_FEATURES].astype('float64')
df_train.describe()

```

	age	fnlwgt	education_num	capital_gain	capital_lo
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000

Puede comprobar el recuento de valores únicos de las entidades native_country. Se puede ver que sólo un hogar proviene de Holanda-Países Bajos. Esta casa no nos traerá ninguna información, pero lo hará a través de un error durante el entrenamiento.

```
df_train.native_country.value_counts()
```

United-States	29170
Mexico	643
?	583
Philippines	198
Germany	137
Canada	121
Puerto-Rico	114
El-Salvador	106
India	100
Cuba	95
England	90
Jamaica	81
South	80
China	75
Italy	73

Dominican-Republic	70
Vietnam	67
Guatemala	64
Japan	62
Poland	60
Columbia	59
Taiwan	51
Haiti	44
Iran	43
Portugal	37
Nicaragua	34
Peru	31
France	29
Greece	29
Ecuador	28
Ireland	24
Hong	20
Cambodia	19
Trinidad&Tobago	19
Thailand	18
Laos	18
Yugoslavia	16
Outlying-US(Guam-USVI-etc)	14
Honduras	13
Hungary	13
Scotland	12
Holland-Netherlands	1
Name: native_country, dtype: int64	

Puede excluir esta fila no informativa del conjunto de datos

```
## Drop Netherland, because only one row
df_train = df_train[df_train.native_country != "Holland-
Netherlands"]
```

A continuación, almacena la posición de las entidades continuas en una lista. Lo necesitará en el siguiente paso para construir la tubería.

El siguiente código recorrerá todos los nombres de columnas en CONTI_Features y obtendrá su ubicación (es decir, su número) y luego lo anexará a una lista llamada conti_features

```
## Get the column index of the categorical features
conti_features = []
for i in CONTI_FEATURES:
    position = df_train.columns.get_loc(i)
    conti_features.append(position)
print(conti_features)
```

```
[0, 2, 10, 4, 11, 12]
```

El siguiente código hace el mismo trabajo que el anterior, pero para la variable categórica. El siguiente código repite lo que ha hecho anteriormente, excepto con las características categóricas.

```
## Get the column index of the categorical features
categorical_features = []
for i in CATE_FEATURES:
    position = df_train.columns.get_loc(i)
    categorical_features.append(position)
print(categorical_features)
```

```
[1, 3, 5, 6, 7, 8, 9, 13]
```

Puede echar un vistazo al conjunto de datos. Tenga en cuenta que, cada entidad categórica es una cadena. No se puede alimentar un modelo con un valor de cadena. Necesita transformar el dataset usando una variable ficticia.

```
df_train.head(5)
```

De hecho, debe crear una columna para cada grupo de la entidad. En primer lugar, puede ejecutar el siguiente código para calcular la cantidad total de columnas necesarias.

```
print(df_train[CATE_FEATURES].nunique(),
      'There are', sum(df_train[CATE_FEATURES].nunique()), 'groups
in the whole dataset')
```

workclass	9
education	16
marital	7

```

occupation      15
relationship     6
race            5
sex              2
native_country   41
dtype: int64 There are 101 groups in the whole dataset

```

Todo el conjunto de datos contiene 101 grupos como se muestra arriba. Por ejemplo, las características de la clase trabajadora tienen nueve grupos. Puede visualizar el nombre de los grupos con los siguientes códigos

`unique ()` devuelve los valores únicos de las entidades categóricas.

```

for i in CATE_FEATURES:
    print(df_train[i].unique())

```

```

['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov'
 '?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college'
 'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-
absent'
 'Separated' 'Married-AF-spouse' 'Widowed']
['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-
specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-
relative']
['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']
['Male' 'Female']
['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand'
 'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-
Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'

```

```
'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinadad&Tobago' 'Greece'  
'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary']
```

Por lo tanto, el conjunto de datos de capacitación contendrá 101 + 7 columnas. Las últimas siete columnas son las características continuas.

SciKit-learn puede hacerse cargo de la conversión. Se hace en dos pasos:

- Primero, debe convertir la cadena a ID. Por ejemplo, State-gov tendrá el ID 1, Self-emp-not-INC ID 2 y así sucesivamente. La función LabelEncoder hace esto por usted
- Transponga cada ID a una nueva columna. Como se mencionó anteriormente, el conjunto de datos tiene el ID del grupo 101. Por lo tanto, habrá 101 columnas que capturarán todos los grupos de entidades categóricas. Scikit-learn tiene una función llamada OneHotEncoder que realiza esta operación

Paso 2) Crear el conjunto de tren/prueba

Ahora que el conjunto de datos está listo, podemos dividirlo 80/20. 80 por ciento para el conjunto de entrenamiento y 20 por ciento para el conjunto de pruebas.

Puede usar `train_test_split`. El primer argumento es que el marco de datos es las entidades y el segundo argumento es el marco de datos de etiqueta. Puede especificar el tamaño del conjunto de pruebas con `test_size`.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df_train[features],
                  df_train.label,
                  test_size =
0.2,
                  random_state=0)
X_train.head(5)
print(X_train.shape, X_test.shape)
```

```
(26048, 14) (6512, 14)
```

Paso 3) Construir la tubería

La canalización facilita la alimentación del modelo con datos consistentes. La idea detrás es poner los datos sin procesar en una 'canalización' para realizar operaciones. Por ejemplo, con el dataset actual, debe estandarizar las variables continuas y convertir los datos categóricos. Tenga en cuenta que puede realizar cualquier operación dentro de la canalización. Por ejemplo, si tiene 'NA' en el conjunto de datos, puede reemplazarlos por la media o mediana. También puede crear nuevas variables.

Usted tiene la opción; codificar los dos procesos o crear una canalización. La primera opción puede conducir a la fuga de datos y crear inconsistencias con el tiempo. Una mejor opción es usar la tubería.

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
```

La tubería realizará dos operaciones antes de alimentar el clasificador logístico:

1. Estandarizar la variable: `StandardScaler ()`
2. Convierta las características categóricas: OneHotEncoder (sparse = False)

Puede realizar los dos pasos utilizando `make_column_transformer`. Esta función no está disponible en la versión actual de scikit-learn (0.19). No es posible con la versión actual realizar el codificador de etiquetas y un codificador activo en la tubería. Es una de las razones por las que

decidimos usar la versión del desarrollador.

`make_column_transformer` es fácil de usar. Debe definir qué columnas aplicar la transformación y qué transformación operar. Por ejemplo, para estandarizar la función continua, puede hacer lo siguiente:

- `conti_features`, `standardScaler()` dentro de `make_column_transformer`.
 - `conti_features`: lista con la variable continua
 - `StandardScaler`: estandarizar la variable

El objeto `OneHotEncoder` dentro `make_column_transformer` codifica automáticamente la etiqueta.

```
preprocess = make_column_transformer(  
    (conti_features, StandardScaler()),  
    ### Need to be numeric not string to specify columns name  
    (categorical_features, OneHotEncoder(sparse=False))  
)
```

Puede probar si la tubería funciona con `fit_transform`. El conjunto de datos debe tener la siguiente forma: `26048, 107`

```
preprocess.fit_transform(X_train).shape  
  
(26048, 107)
```

El transformador de datos está listo para usar. Puede crear la tubería con `make_pipeline`. Una vez que los datos se transforman, puede alimentar la regresión logística.

```
model = make_pipeline(  
    preprocess,  
    LogisticRegression())
```

El entrenamiento de un modelo con scikit-learn es trivial. Debe usar el ajuste del objeto precedido por la tubería, es decir, el modelo. Puede

imprimir la precisión con el objeto de partitura desde la biblioteca scikit-learn

```
model.fit(X_train, y_train)
print("logistic regression score: %f" % model.score(X_test,
y_test))
```

```
logistic regression score: 0.850891
```

Finalmente, puede predecir las clases con predict_proba. Devuelve la probabilidad de cada clase. Tenga en cuenta que se suma a uno.

```
model.predict_proba(X_test)
```

```
array([[0.83576663, 0.16423337],
       [0.94582765, 0.05417235],
       [0.64760587, 0.35239413],
       ...,
       [0.99639252, 0.00360748],
       [0.02072181, 0.97927819],
       [0.56781353, 0.43218647]])
```

Paso 4) Uso de nuestra tubería en una búsqueda de cuadrícula

El ajuste del hiperparámetro (variables que determinan la estructura de la red como unidades ocultas) puede ser tedioso y agotador. Una forma de evaluar el modelo podría ser cambiar el tamaño del conjunto de entrenamiento y evaluar los resultados. Puede repetir este método diez veces para ver las métricas de puntuación. Sin embargo, es demasiado trabajo.

En su lugar, scikit-learn proporciona una función para llevar a cabo el ajuste de parámetros y la validación cruzada.

Validación cruzada

Validación cruzada significa durante el entrenamiento, el conjunto de entrenamiento es dividido en n número de veces en pliegues y luego evalúa el modelo n veces. Por ejemplo, si `cv` se establece en `10`, el conjunto de entrenamiento se capacita y se evalúa diez veces. En cada ronda, el clasificador elige al azar nueve veces para entrenar el modelo, y el décimo pliegue está destinado a la evaluación.

Búsqueda de cuadrícula Cada clasificador tiene hiperparámetros para ajustar. Puede probar valores diferentes o puede establecer una cuadrícula de parámetros. Si vas al sitio web oficial de scikit-learn, puedes ver que el clasificador logístico tiene diferentes parámetros para ajustar. Para que el entrenamiento sea más rápido, elige ajustar el parámetro `C`. Controla el parámetro de regularización. Debería ser positivo. Un valor pequeño da más peso al regularizador.

Puede utilizar el objeto `GridSearchCV`. Debe crear un diccionario que

contenga los hiperparámetros que desea ajustar.

Enumerar los hiperparámetros seguidos de los valores que desea probar. Por ejemplo, para ajustar el parámetro C, usa:

- 'logisticregression__C': [0.1, 1.0, 1.0]: El parámetro va precedido por el nombre, en minúsculas, del clasificador y dos guiones bajos.

El modelo intentará cuatro valores diferentes: 0.001, 0.01, 0.1 y 1.

Entrena el modelo usando 10 pliegues: cv = 10

```
from sklearn.model_selection import GridSearchCV
# Construct the parameter grid
param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1.0],}
```

Puede entrenar el modelo usando GridSearchCV con los parámetros gri y cv.

```
# Train the model
grid_clf = GridSearchCV(model,
                        param_grid,
                        cv=10,
                        iid=False)
grid_clf.fit(X_train, y_train)
```

SALIDA

```
GridSearchCV(cv=10, error_score='raise-deprecating',
            estimator=Pipeline(memory=None,
            steps=[('columntransformer', ColumnTransformer(n_jobs=1,
            remainder='drop', transformer_weights=None,
            transformers=[('standardscaler', StandardScaler(copy=True,
            with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),
            ('onehotencoder', OneHotEncoder(categorical_features=None,
            categories=None,...ty='l2', random_state=None, solver='liblinear',
            tol=0.0001,
            verbose=0, warm_start=False))]]),
```

```
    fit_params=None, iid=False, n_jobs=1,
    param_grid={'logisticregression__C': [0.001, 0.01, 0.1,
1.0]},
    pre_dispatch='2*n_jobs', refit=True,
return_train_score='warn',
    scoring=None, verbose=0)
```

Para acceder a los mejores parámetros, utilice `best_params_`

```
grid_clf.best_params_
```

SALIDA

```
{'logisticregression__C': 1.0}
```

Después de entrenar el modelo con cuatro valores de regularización diferentes, el parámetro óptimo es

```
print("best logistic regression from grid search: %f" %
grid_clf.best_estimator_.score(X_test, y_test))
```

mejor regresión logística desde la búsqueda de cuadrícula: 0.850891

Para acceder a las probabilidades previstas:

```
grid_clf.best_estimator_.predict_proba(X_test)
```

```
array([[0.83576677, 0.16423323],
       [0.9458291 , 0.0541709 ],
       [0.64760416, 0.35239584],
       ...,
       [0.99639224, 0.00360776],
       [0.02072033, 0.97927967],
       [0.56782222, 0.43217778]])
```

Modelo XGBoost con scikit-learn

Vamos a tratar de entrenar a uno de los mejores clasificadores en el mercado. XgBoost es una mejora sobre el bosque aleatorio. El fondo teórico del clasificador fuera del alcance de este tutorial. Tenga en cuenta que, XgBoost ha ganado muchas competiciones kaggle. Con un tamaño promedio de dataset, puede funcionar tan bien como un algoritmo de aprendizaje profundo o incluso mejor.

El clasificador es un reto para entrenar porque tiene un gran número de parámetros para ajustar. Por supuesto, puede usar GridSearchCV para elegir el parámetro por usted.

En su lugar, veamos cómo usar una mejor manera de encontrar los parámetros óptimos. GridSearchCV puede ser tedioso y muy largo para entrenar si pasa muchos valores. El espacio de búsqueda crece junto con el número de parámetros. Una solución preferible es usar RandomizedSearchCV. Este método consiste en elegir los valores de cada hiperparámetro después de cada iteración aleatoriamente. Por ejemplo, si el clasificador está entrenado más de 1000 iteraciones, entonces se evalúan 1000 combinaciones. Funciona más o menos como.

GridSearchCV

Si la biblioteca no está instalada, utilice pip3 install xgboost o

```
use import sys  
!{sys.executable} -m pip install xgboost
```

En el entorno de Jupyter

Próximo,

```
import xgboost
```

```
from sklearn.model_selection import RandomizedSearchCV  
from sklearn.model_selection import StratifiedKFold
```

El siguiente paso incluye especificar los parámetros que desea ajustar. Puede consultar la documentación oficial para ver todos los parámetros a ajustar. Por el bien del tutorial, sólo puede elegir dos hiperparámetros con dos valores cada uno. XgBoost tarda mucho tiempo en entrenar, cuantos más hiperparámetros haya en la cuadrícula, más tiempo tendrá que esperar.

```
params = {  
    'xgbclassifier__gamma': [0.5, 1],  
    'xgbclassifier__max_depth': [3, 4]  
}
```

Construye una nueva canalización con el clasificador XgBoost. Elija definir 600 estimadores. Tenga en cuenta que n_estimators son un parámetro que se puede ajustar. Un valor alto puede conducir a un ajuste excesivo. Puede probar por sí mismo diferentes valores, pero tenga en cuenta que puede tomar horas. Se utiliza el valor predeterminado para los demás parámetros

```
model_xgb = make_pipeline(  
    preprocess,  
    xgboost.XGBClassifier(  
        n_estimators=600,  
        objective='binary:logistic',  
        silent=True,  
        nthread=1)  
)
```

Puede mejorar la validación cruzada con el validador cruzado de K-Folds estratificados. Usted construye sólo tres pliegues aquí para acelerar el cálculo pero reducir la calidad. Aumente este valor a 5 o 10 en casa para mejorar los resultados.

Elija entrenar el modelo en cuatro iteraciones.

```
skf = StratifiedKFold(n_splits=3,
                      shuffle = True,
                      random_state = 1001)

random_search = RandomizedSearchCV(model_xgb,
                                    param_distributions=params,
                                    n_iter=4,
                                    scoring='accuracy',
                                    n_jobs=4,
                                    cv=skf.split(X_train, y_train),
                                    verbose=3,
                                    random_state=1001)
```

La búsqueda aleatoria está lista para usar, puede entrenar el modelo

```
#grid_xgb = GridSearchCV(model_xgb, params, cv=10, iid=False)
random_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8759645283888057, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8729701715996775, total= 1.0min
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8706519235199263, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8735460094437406, total= 1.3min
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
```

```
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8722791661868018, total= 57.7s
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8753886905447426, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8697304768486523, total= 1.3min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8740066797189912, total= 1.4min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8707671043538355, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8729701715996775, total= 1.2min
[Parallel(n_jobs=4)]: Done 10 out of 12 | elapsed: 3.6min
remaining: 43.5s
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8736611770125533, total= 1.2min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8692697535130154, total= 1.2min
```

```
[Parallel(n_jobs=4)]: Done 12 out of 12 | elapsed: 3.6min
finished
/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:737:
DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24.
This will change numeric results when test-set sizes are unequal.
DeprecationWarning)
```

```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at
0x1101eb830>,
                  error_score='raise-deprecating',
                  estimator=Pipeline(memory=None,
                                     steps=[('columntransformer', ColumnTransformer(n_jobs=1,
                                         remainder='drop', transformer_weights=None,
                                         transformers=[('standardscaler', StandardScaler(copy=True,
                                         with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),
                                         ('onehotencoder', OneHotEncoder(categorical_features=None,
```

```
categories=None, ...
    reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
    silent=True, subsample=1))],
    fit_params=None, iid='warn', n_iter=4, n_jobs=4,
    param_distributions={'xgbclassifier_gamma': [0.5, 1],
'xgbclassifier_max_depth': [3, 4]},
    pre_dispatch='2*n_jobs', random_state=1001, refit=True,
    return_train_score='warn', scoring='accuracy', verbose=3)
```

Como puede ver, XgBoost tiene una puntuación mejor que la regresión lógica anterior.

```
print("Best parameter", random_search.best_params_)
print("best logistic regression from grid search: %f" %
random_search.best_estimator_.score(X_test, y_test))
```

```
Best parameter {'xgbclassifier_max_depth': 3,
'xgbclassifier_gamma': 0.5}
best logistic regression from grid search: 0.873157
```

```
random_search.best_estimator_.predict(X_test)
```

```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '>50K', '<=50K'],
dtype=object)
```

Crear DNN con MLPClassifier en scikit-learn

Finalmente, puedes entrenar un algoritmo de aprendizaje profundo con scikit-learn. El método es el mismo que el otro clasificador. El clasificador está disponible en MLPClassifier.

```
from sklearn.neural_network import MLPClassifier
```

Defina el siguiente algoritmo de aprendizaje profundo:

- Adam solucionador
- Función de activación de Relu
- Alfa = 0.0001
- tamaño del lote de 150
- Dos capas ocultas con 100 y 50 neuronas respectivamente

```
model_dnn = make_pipeline(  
    preprocess,  
    MLPClassifier(solver='adam',  
                  alpha=0.0001,  
                  activation='relu',  
                  batch_size=150,  
                  hidden_layer_sizes=(200, 100),  
                  random_state=1))
```

Puede cambiar el número de capas para mejorar el modelo

```
model_dnn.fit(X_train, y_train)  
print("DNN regression score: %f" % model_dnn.score(X_test,  
y_test))
```

Puntuación de regresión DNN: 0.821253

LIME: Confía en tu modelo

Ahora que tienes un buen modelo, necesitas una herramienta para confiar en él. Algoritmo de aprendizaje automático, especialmente bosque aleatorio y red neuronal, son conocidos por ser algoritmo black-box. Decir de otra manera, funciona pero nadie sabe por qué.

Tres investigadores han encontrado una gran herramienta para ver cómo la computadora hace una predicción. El periódico se llama *¿Por qué debería confiar en ti?*

Desarrollaron un algoritmo llamado **Explicaciones agnósticas de modelos interpretables locales (LIME)**.

Tomemos un ejemplo:

a veces no sabes si puedes confiar en una predicción de aprendizaje automático:

Un médico, por ejemplo, no puede confiar en un diagnóstico sólo porque una computadora lo dijo. También necesita saber si puede confiar en el modelo antes de ponerlo en producción.

Imagine que podemos entender por qué cualquier clasificador está haciendo una predicción incluso modelos increíblemente complicados como redes neuronales, bosques aleatorios o svms con cualquier núcleo

será más accesible para confiar en una predicción si podemos entender las razones detrás de ella. A partir del ejemplo con el médico, si el modelo le dijo qué síntomas son esenciales confiaría en él, también es más fácil averiguar si no debe confiar en el modelo.

Lime puede decirle qué características afectan las decisiones del

clasificador

Preparación de datos

Son un par de cosas que debes cambiar para ejecutar LIME con python. En primer lugar, debe instalar cal en el terminal. Puede usar pip install lime

Lime hace uso del objeto LimetabularExplainer para aproximar el modelo localmente. Este objeto requiere:

- un conjunto de datos en formato numpy
- El nombre de las entidades: feature_names
- El nombre de las clases: class_names
- El índice de la columna de las entidades categóricas: categorical_features
- El nombre del grupo para cada elemento categórico: categorical_names

Crear conjunto de trenes numpy

Puede copiar y convertir df_train de pandas a numpy muy fácilmente

```
df_train.head(5)
# Create numpy data
df_lime = df_train
df_lime.head(3)
```

Obtener el nombre de la clase La etiqueta es accesible con el objeto unique (). Debería ver:

- '<=50K'
- '>50K'

```
# Get the class name
class_names = df_lime.label.unique()
```

```
class_names
```

```
array(['<=50K', '>50K'], dtype=object)
```

índice de la columna de las características categóricas

Puede utilizar el método que se inclina antes para obtener el nombre del grupo. Codificar la etiqueta con LabelEncoder. Repita la operación en todas las entidades categóricas.

```
##  
import sklearn.preprocessing as preprocessing  
categorical_names = {}  
for feature in CATE_FEATURES:  
    le = preprocessing.LabelEncoder()  
    le.fit(df_lime[feature])  
    df_lime[feature] = le.transform(df_lime[feature])  
    categorical_names[feature] = le.classes_  
print(categorical_names)  
  
{'workclass': array(['?', 'Federal-gov', 'Local-gov', 'Never-  
worked', 'Private',  
        'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-  
pay'],  
        dtype=object), 'education': array(['10th', '11th', '12th',  
'1st-4th', '5th-6th', '7th-8th', '9th',  
        'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-  
grad',  
        'Masters', 'Preschool', 'Prof-school', 'Some-college'],  
        dtype=object), 'marital': array(['Divorced', 'Married-AF-  
spouse', 'Married-civ-spouse',  
        'Married-spouse-absent', 'Never-married', 'Separated',  
'Widowed'],  
        dtype=object), 'occupation': array(['?', 'Adm-clerical',  
'Armed-Forces', 'Craft-repair',  
        'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',  
        'Machine-op-inspct', 'Other-service', 'Priv-house-serv',  
        'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-  
support',  
        'Transport-moving'], dtype=object), 'relationship':  
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',  
'Unmarried', 'Wife'], dtype=object), 'race': array(['Amer-
```

```
Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',
       'White'], dtype=object), 'sex': array(['Female', 'Male'],
dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',
'China', 'Columbia', 'Cuba',
'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',
'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',
'Honduras',
'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',
'Jamaica',
'Japan', 'Laos', 'Mexico', 'Nicaragua',
'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',
'Poland',
'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',
'Thailand', 'Trinadad&Tobago', 'United-States', 'Vietnam',
'Yugoslavia'], dtype=object)}
```

```
df_lime.dtypes
```

age	float64
workclass	int64
fnlwgt	float64
education	int64
education_num	float64
marital	int64
occupation	int64
relationship	int64
race	int64
sex	int64
capital_gain	float64
capital_loss	float64
hours_week	float64
native_country	int64
label	object
dtype:	object

Ahora que el dataset está listo, puede construir el dataset diferente. Realmente transforma los datos fuera de la canalización para evitar errores con LIME. El entrenamiento establecido en el LimetabularExplainer debe ser un array numpy sin cadena. Con el método anterior, tiene un conjunto de datos de formación ya convertido.

```
from sklearn.model_selection import train_test_split
X_train_lime, X_test_lime, y_train_lime, y_test_lime =
```

```
train_test_split(df_lime[features],  
                           df_lime.label,  
                           test_size =  
                           0.2,  
                           random_state=0)  
X_train_lime.head(5)
```

Puede hacer la tubería con los parámetros óptimos de XgBoost

```
model_xgb = make_pipeline(  
    preprocess,  
    xgboost.XGBClassifier(max_depth = 3,  
                           gamma = 0.5,  
                           n_estimators=600,  
                           objective='binary:logistic',  
                           silent=True,  
                           nthread=1))  
  
model_xgb.fit(X_train_lime, y_train_lime)
```

```
/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-  
packages/sklearn/preprocessing/_encoders.py:351: FutureWarning: The  
handling of integer data will change in version 0.22. Currently,  
the categories are determined based on the range [0, max(values)],  
while in the future they will be determined based on the unique  
values.
```

If you want the future behavior and silence this warning, you can specify "categories='auto'." In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```
Pipeline(memory=None,  
         steps=[('columntransformer', ColumnTransformer(n_jobs=1,  
                                         remainder='drop', transformer_weights=None,  
                                         transformers=[('standardscaler', StandardScaler(copy=True,  
                                                 with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12]),  
                                         ('onehotencoder', OneHotEncoder(categorical_features=None,  
                                             categories=None,...  
                                             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,  
                                             silent=True, subsample=1))])]
```

Recibéis una advertencia. La advertencia explica que no es necesario

crear un codificador de etiquetas antes de la canalización. Si no desea utilizar LIME, puede utilizar el método de la primera parte del tutorial. De lo contrario, puede mantener con este método, primero crear un dataset codificado, establecer obtener el codificador de uno activo dentro de la tubería.

```
print("best logistic regression from grid search: %f" %  
model_xgb.score(X_test_lime, y_test_lime))
```

```
best logistic regression from grid search: 0.873157
```

```
model_xgb.predict_proba(X_test_lime)
```

```
array([[7.9646105e-01, 2.0353897e-01],  
       [9.5173013e-01, 4.8269872e-02],  
       [7.9344827e-01, 2.0655173e-01],  
       ...,  
       [9.9031430e-01, 9.6856682e-03],  
       [6.4581633e-04, 9.9935418e-01],  
       [9.7104281e-01, 2.8957171e-02]], dtype=float32)
```

Antes de usar LIME en acción, vamos a crear una matriz numpy con las características de la clasificación incorrecta. Puede usar esa lista más tarde para tener una idea sobre lo que confunde al clasificador.

```
temp = pd.concat([X_test_lime, y_test_lime], axis= 1)  
temp['predicted'] = model_xgb.predict(X_test_lime)  
temp['wrong']= temp['label'] != temp['predicted']  
temp = temp.query('wrong==True').drop('wrong', axis=1)  
temp= temp.sort_values(by=['label'])  
temp.shape
```

(826, 16)

Se crea una función lambda para recuperar la predicción del modelo con los nuevos datos. Lo necesitarás pronto.

```
predict_fn = lambda x: model_xgb.predict_proba(x).astype(float)  
X_test_lime.dtypes
```

```
age           float64
workclass     int64
fnlwgt        float64
education     int64
education_num float64
marital       int64
occupation    int64
relationship   int64
race          int64
sex           int64
capital_gain  float64
capital_loss  float64
hours_week    float64
native_country int64
dtype: object
```

```
predict_fn(X_test_lime)
```

```
array([[7.96461046e-01,  2.03538969e-01],
       [9.51730132e-01,  4.82698716e-02],
       [7.93448269e-01,  2.06551731e-01],
       ...,
       [9.90314305e-01,  9.68566816e-03],
       [6.45816326e-04,  9.99354184e-01],
       [9.71042812e-01,  2.89571714e-02]])
```

Convierte el marco de datos pandas a numpy array

```
X_train_lime = X_train_lime.values
X_test_lime = X_test_lime.values
X_test_lime
```

```
array([[4.00000e+01,  5.00000e+00,  1.93524e+05,  ...,  0.00000e+00,
       4.00000e+01,  3.80000e+01],
       [2.70000e+01,  4.00000e+00,  2.16481e+05,  ...,  0.00000e+00,
       4.00000e+01,  3.80000e+01],
       [2.50000e+01,  4.00000e+00,  2.56263e+05,  ...,  0.00000e+00,
       4.00000e+01,  3.80000e+01],
       ...,
       [2.80000e+01,  6.00000e+00,  2.11032e+05,  ...,  0.00000e+00,
       4.00000e+01,  2.50000e+01],
       [4.40000e+01,  4.00000e+00,  1.67005e+05,  ...,  0.00000e+00,
       6.00000e+01,  3.80000e+01],
       [5.30000e+01,  4.00000e+00,  2.57940e+05,  ...,  0.00000e+00,
```

```

    4.00000e+01, 3.80000e+01]]))

model_xgb.predict_proba(X_test_lime)

array([[7.9646105e-01, 2.0353897e-01],
       [9.5173013e-01, 4.8269872e-02],
       [7.9344827e-01, 2.0655173e-01],
       ...,
       [9.9031430e-01, 9.6856682e-03],
       [6.4581633e-04, 9.9935418e-01],
       [9.7104281e-01, 2.8957171e-02]], dtype=float32)

print(features,
      class_names,
      categorical_features,
      categorical_names)

['age', 'workclass', 'fnlwgt', 'education', 'education_num',
'marital', 'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss', 'hours_week', 'native_country']
['<=50K' '>50K'] [1, 3, 5, 6, 7, 8, 9, 13] {'workclass':
array(['?', 'Federal-gov', 'Local-gov', 'Never-worked', 'Private',
      'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-
pay'],
      dtype=object), 'education': array(['10th', '11th', '12th',
'1st-4th', '5th-6th', '7th-8th', '9th',
      'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-
grad',
      'Masters', 'Preschool', 'Prof-school', 'Some-college'],
      dtype=object), 'marital': array(['Divorced', 'Married-AF-
spouse',
      'Married-civ-spouse',
      'Married-spouse-absent', 'Never-married', 'Separated',
      'Widowed'],
      dtype=object), 'occupation': array(['?', 'Adm-clerical',
'Armed-Forces', 'Craft-repair',
      'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',
      'Machine-op-inspct', 'Other-service', 'Priv-house-serv',
      'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-
support',
      'Transport-moving'],
      dtype=object), 'relationship':
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',
      'Unmarried', 'Wife'],
      dtype=object), 'race': array(['Amer-
Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',
      'White'],
      dtype=object), 'sex': array(['Female', 'Male'],
      dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',
      'England', 'Greece', 'Holand-Netherlands', 'Portugal', 'South Africa',
      'United States', 'United Kingdom'],
      dtype=object)]

```

```
'China', 'Columbia', 'Cuba',
'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',
'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',
'Honduras',
'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',
'Jamaica',
'Japan', 'Laos', 'Mexico', 'Nicaragua',
'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',
'Poland',
'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',
'Thailand', 'Trinadad&Tobago', 'United-States', 'Vietnam',
'Yugoslavia'], dtype=object)}
```

```
import lime
import lime.lime_tabular
### Train should be label encoded not one hot encoded
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_lime ,
                                                    feature_names =
features,
class_names=class_names,
categorical_features=categorical_features,
categorical_names=categorical_names,
                                                    kernel_width=3)
```

Vamos a elegir un hogar aleatorio del conjunto de pruebas y ver la predicción del modelo y cómo el ordenador hizo su elección.

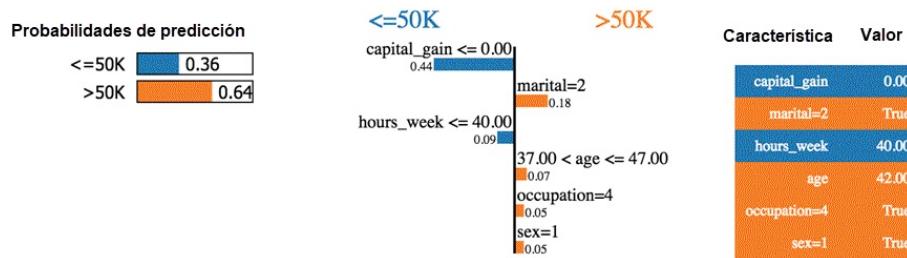
```
import numpy as np
np.random.seed(1)
i = 100
print(y_test_lime.iloc[i])
>50K
```

```
X_test_lime[i]
```

```
array([4.20000e+01, 4.00000e+00, 1.76286e+05, 7.00000e+00,
1.20000e+01,
2.00000e+00, 4.00000e+00, 0.00000e+00, 4.00000e+00,
1.00000e+00,
0.00000e+00, 0.00000e+00, 4.00000e+01, 3.80000e+01])
```

Puede usar el explicador con `explain_instance` para verificar la explicación detrás del modelo

```
exp = explainer.explain_instance(X_test_lime[i], predict_fn,  
num_features=6)  
exp.show_in_notebook(show_all=False)
```



Podemos ver que el clasificador predijo correctamente el hogar. El ingreso es, de hecho, superior a los 50k.

Lo primero que podemos decir es que el clasificador no está tan seguro sobre las probabilidades predecidas. La máquina predice que el hogar tiene un ingreso superior a 50k con una probabilidad de 64%. Este 64% se compone de ganancia de capital y conyugal. El color azul contribuye negativamente a la clase positiva y la línea naranja, positivamente.

El clasificador está confundido porque la ganancia de capital de este hogar es nula, mientras que la ganancia de capital suele ser un buen predictor de riqueza. Además, el hogar trabaja menos de 40 horas por semana. La edad, la ocupación y el género contribuyen positivamente al clasificador.

Si el estado civil fuera soltero, el clasificador habría predicho un ingreso por debajo de 50k ($0.64 - 0.18 = 0.46$)

Podemos intentarlo con otro hogar que ha sido clasificado erróneamente

```
temp.head(3)  
temp.iloc[1, :-2]
```

```

age                      58
workclass                 4
fnlwgt                  68624
education                  11
education_num                9
marital                     2
occupation                   4
relationship                  0
race                        4
sex                          1
capital_gain                  0
capital_loss                  0
hours_week                   45
native_country                  38
Name: 20931, dtype: object

```

```

i = 1
print('This observation is', temp.iloc[i,-2:])

```

```

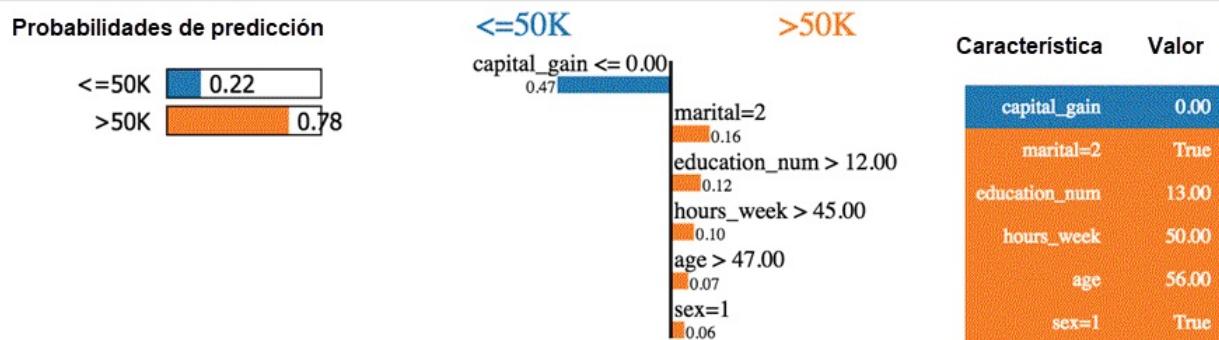
This observation is label      <=50K
predicted        >50K
Name: 20931, dtype: object

```

```

exp = explainer.explain_instance(temp.iloc[1,:-2], predict_fn,
num_features=6)
exp.show_in_notebook(show_all=False)

```



El clasificador predijo un ingreso por debajo de 50k mientras que es falso. Esta casa parece extraña. No tiene una ganancia de capital, ni pérdida de capital. Él está divorciado y tiene 60 años, y es un pueblo educado, es decir, education_num > 12. De acuerdo con el patrón general, este hogar debería, como explica el clasificador, obtener un

ingreso por debajo de los 50k.

Tratas de jugar con LIME. Notarás errores graves por parte del clasificador.

Puede verificar el GitHub del propietario de la biblioteca. Proporcionan documentación adicional para la clasificación de imágenes y textos.

Resumen

A continuación se muestra una lista de algunos comandos útiles con scikit learn version > = 0.20

crear conjunto de datos de tren/prueba	alumnos divididos
Crear una canalización	
seleccione la columna y aplique la transformación	transformador de columna
tipo de transformación	
estandarizar	StandardScaler
min max	MinMaxScaler
Normalizar	Normalizador
Imputar valor faltante	Imputador
Convertir categórica	OneHotEncoder
Ajustar y transformar los datos	fit_transform
Hacer la tubería	make_pipeline
Modelo básico	

regresión logística	Regresión logística
XgBoost	XGBClassifier
Red neuronal	Clasificador MLP
Búsqueda de cuadrícula	GridSearchCV
Búsqueda aleatoria	RandomizedSearchCV

Capítulo 13: Regresión lineal

Regresión lineal

En este tutorial, aprenderá los principios básicos de regresión lineal y aprendizaje automático en general.

TensorFlow proporciona herramientas para tener un control total de los cálculos. Esto se hace con la API de bajo nivel. Además de eso, TensorFlow está equipado con una amplia gama de API para realizar muchos algoritmos de aprendizaje automático. Esta es la API de alto nivel. TensorFlow los llama estimadores

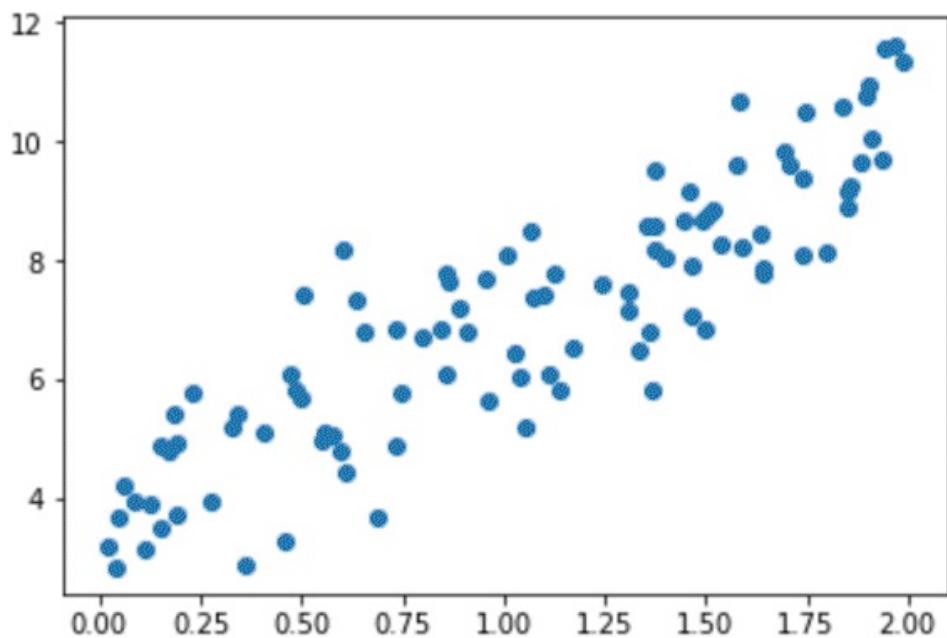
- API de bajo nivel: Construye la arquitectura, optimización del modelo desde cero. Es complicado para un principiante
- API de alto nivel: Defina el algoritmo. Es fácil de usar. TensorFlow proporciona una caja de herramientas de llamadas **estimador** para construir, entrenar, evaluar y hacer una predicción.

En este tutorial, utilizará el método **sólo estimadores**. Los cálculos son más rápidos y son más fáciles de implementar. La primera parte del tutorial explica cómo utilizar el optimizador de descenso de gradiente para entrenar una regresión lineal. En una segunda parte, utilizará el dataset de Boston para predecir el precio de una casa utilizando el estimador TensorFlow.

Cómo entrenar un modelo de regresión lineal

Antes de comenzar a entrenar el modelo, echemos un vistazo a lo que es una regresión lineal.

Imagine que tiene dos variables, x e y y su tarea es predecir el valor de conocer el valor de. Si traza los datos, puede ver una relación positiva entre la variable independiente, x y la variable dependiente y.



Usted puede observar, si $x = 1$, y será aproximadamente igual a 6 y si $x = 2$, y estará alrededor de 8.5.

Este no es un método muy preciso y propenso a errores, especialmente con un conjunto de datos con cientos de miles de puntos.

Una regresión lineal se evalúa con una ecuación. La variable y se explica por una o varias covariables. En su ejemplo, solo hay una variable

dependiente. Si tienes que escribir esta ecuación, será:

$$y = \beta + \alpha X + \epsilon$$

Con:

- β es el sesgo. es decir, si $x = 0$, $y = \beta$
- α es el peso asociado a x
- ϵ es el residual o el error del modelo. Incluye lo que el modelo no puede aprender de los datos

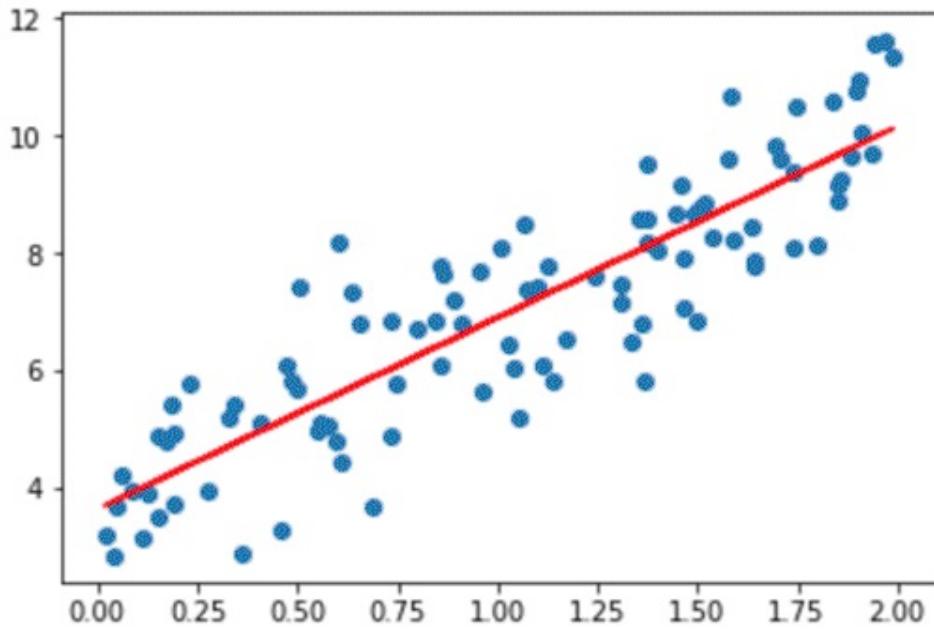
Imagine que se ajusta al modelo y encuentra la siguiente solución para:

- $\beta = 3.8$
- $\alpha = 2.78$

Puede sustituir esos números en la ecuación y se convierte en:

$$y = 3.8 + 2.78x$$

Ahora tiene una mejor manera de encontrar los valores de y . Es decir, puede reemplazar x por cualquier valor que desee predecir y . En la siguiente imagen, hemos reemplazado x en la ecuación con todos los valores del dataset y trazar el resultado.



La línea roja representa el valor ajustado, es decir, los valores de y para cada valor de x . No necesita ver el valor de x para predecir y , para cada x hay alguno que pertenece a la línea roja. También puede predecir para valores de x superior a 2!

Si desea extender la regresión lineal a más covariables, puede agregar más variables al modelo. La diferencia entre el análisis tradicional y la regresión lineal es que la regresión lineal mira cómo y reaccionará para cada variable x tomada de forma independiente.

Veamos un ejemplo. Imagina que quieres predecir las ventas de una heladería. El conjunto de datos contiene información diferente, como el clima (es decir, lluvioso, soleado, nublado), información del cliente (es decir, salario, género, estado civil).

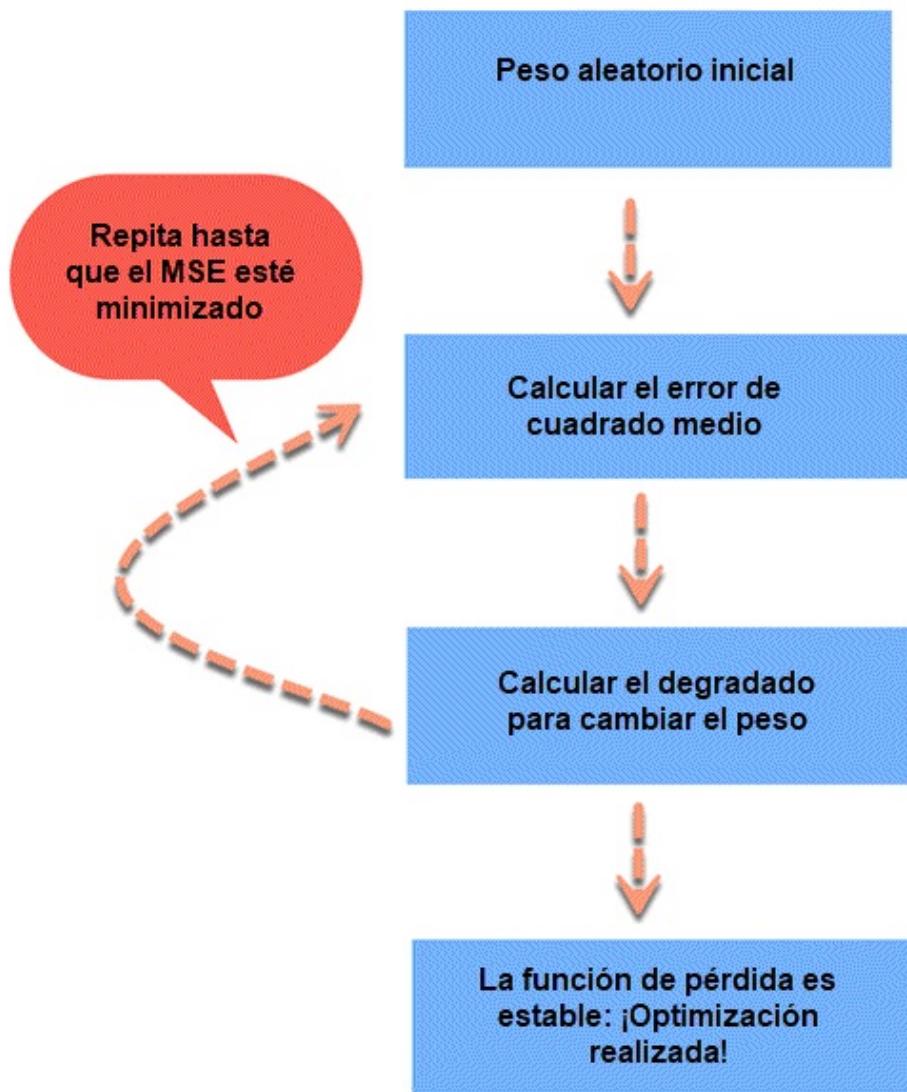
El análisis tradicional intentará predecir la venta calculando el promedio de cada variable y tratar de estimar la venta para diferentes escenarios. Esto conducirá a predicciones malas y restringirá el análisis al escenario elegido.

Si utiliza regresión lineal, puede escribir esta ecuación:

$$Sales = \beta + \alpha_1 weather + \alpha_2 salary + \alpha_3 gender + \alpha_4 marital + \epsilon$$

El algoritmo encontrará la mejor solución para los pesos; significa que intentará minimizar el costo (la diferencia entre la línea ajustada y los puntos de datos).

Cómo funciona el algoritmo



El algoritmo elegirá un número aleatorio para cada β y α y reemplace el valor de x para obtener el valor previsto de y. Si el conjunto de datos tiene 100 observaciones, el algoritmo calcula 100 valores predichos.

Podemos calcular el error, señaló ϵ del modelo, que es la diferencia entre el valor predicho y el valor real. Un error positivo significa que el modelo subestima la predicción de y, y un error negativo significa que el modelo sobreestima la predicción de y.

$$\epsilon = y - y_{pred}$$

Su objetivo es minimizar el cuadrado del error. El algoritmo calcula la media del error cuadrado. Este paso se llama minimización del error. Para la regresión lineal es el **Error de cuadrado medio**, también llamado MSE Matemáticamente, es:

$$MSE(X) = \frac{1}{m} \sum_{i=1}^m (\theta^T x^i - y^i)^2$$

¿Dónde:

- θ^T es los pesos por lo que $\theta^T x^i$ se refiere al valor predicho
- y es los valores reales
- m es el número de observaciones

Tenga en cuenta que θ^T significa que utiliza la transposición de las

matrices. $\frac{1}{m} \sum$ es la notación matemática de la media.

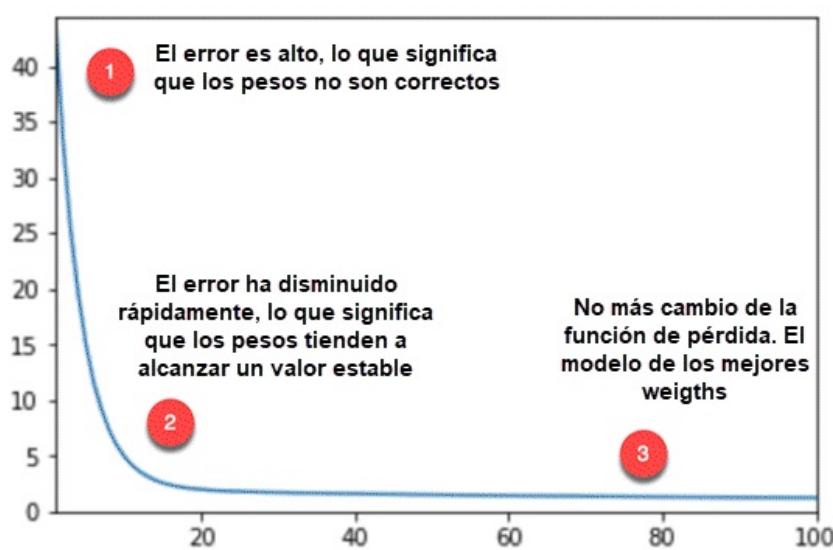
El objetivo es encontrar el mejor θ que minimizan el MSE

Si el error promedio es grande, significa que el modelo funciona mal y los

pesos no se eligen correctamente. Para corregir los pesos, debe usar un optimizador. El optimizador tradicional se llama **Descenso degradado**.

El descenso del gradiente toma la derivada y disminuye o aumenta el peso. Si la derivada es positiva, el peso disminuye. Si la derivada es negativa, el peso aumenta. El modelo actualizará los pesos y volverá a calcular el error. Este proceso se repite hasta que el error ya no cambia. Cada proceso se llama un **iteración**. Además, los gradientes se multiplican por una tasa de aprendizaje, lo que indica la velocidad del aprendizaje.

Si la tasa de aprendizaje es demasiado pequeña, tomará mucho tiempo para que el algoritmo converja (es decir, requiere muchas iteraciones). Si la tasa de aprendizaje es demasiado alta, es posible que el algoritmo nunca converja.



Puede ver en la imagen de arriba, el modelo repite el proceso unas 20 veces antes para encontrar un valor estable para los pesos, alcanzando así el error más bajo.

Tenga en cuenta que, el error no es igual a cero, pero se estabiliza alrededor de 5. Significa que el modelo hace un error típico de 5. Si desea reducir el error, debe agregar más información al modelo, como más variables o utilizar diferentes estimadores.

¿Recuerdas la primera ecuación?

$$y = \beta + \alpha X + \epsilon$$

Los pesos finales son 3.8 y 2.78. El siguiente video muestra cómo el descenso de gradiente optimiza la función de pérdida para encontrar estos pesos

Cómo entrenar una Regresión Lineal con TensorFlow

Ahora que tiene una mejor comprensión de lo que está sucediendo detrás del capó, está listo para usar la API del estimador proporcionada por TensorFlow para entrenar su primera regresión lineal.

Utilizará el conjunto de datos de Boston, que incluye las siguientes variables

Crim	Tasa de delincuencia per cápita por ciudad
zn	proporción de terrenos residenciales zonados para lotes de más de 25.000 pies cuadrados.
indus	proporción de hectáreas de negocios no minoristas por ciudad.
nox	concentración de óxidos nítricos
rm	promedio de habitaciones por vivienda
edad	proporción de unidades ocupadas por sus propietarios construidas antes de 1940
dis	distancias ponderadas a cinco centros de empleo de Boston
impuestos	valor total de la propiedad de la tasa de impuesto por dólar 10.000
ptratio	proporción alumnos/maestro por ciudad
medv	Valor medio de las viviendas ocupadas por los propietarios en miles de dólares

Creará tres conjuntos de datos diferentes:

conjunto de datos	objetivo	forma
Entrenamiento	Entrenar el modelo y obtener los pesos	400, 10
Evaluación	Evaluuar el rendimiento del modelo en datos no vistos	100, 10
Predice	Utilice el modelo para predecir el valor de la casa en nuevos datos	6, 10

El objetivo es utilizar las características del conjunto de datos para predecir el valor de la casa.

Durante la segunda parte del tutorial, aprenderá a utilizar TensorFlow con tres formas diferentes de importar los datos:

- Con Pandas
- Con Numpy
- Sólo TF

Tenga en cuenta que, todas las opciones **proporcionan los mismos resultados.**

Aprenderá a usar la API de alto nivel para construir, entrenar y evaluar un modelo de regresión lineal. Si estaba usando la API de bajo nivel, tenía que definir a mano lo siguiente:

- Función de pérdida
- Optimizar: Descenso degradado
- Multiplicación de matrices
- Gráfico y tensor

Esto es tedioso y más complicado para principiantes.

Pandas

Debe importar las bibliotecas necesarias para entrenar el modelo.

```
import pandas as pd  
from sklearn import datasets  
import tensorflow as tf  
import itertools
```

Paso 1) Importar los datos con panda.

Defina los nombres de las columnas y los almacene en COLUMNS. Puede utilizar pd.read_csv () para importar los datos.

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",  
           "dis", "tax", "ptratio", "medv"]
```

```
training_set = pd.read_csv("E:/boston_train.csv",  
                           skipinitialspace=True,skiprows=1, names=COLUMNS)
```

```
test_set = pd.read_csv("E:/boston_test.csv",  
                       skipinitialspace=True,skiprows=1, names=COLUMNS)
```

```
prediction_set = pd.read_csv("E:/boston_predict.csv",  
                            skipinitialspace=True,skiprows=1, names=COLUMNS)
```

Puede imprimir la forma de los datos.

```
print(training_set.shape, test_set.shape, prediction_set.shape)
```

Salida

```
(400, 10) (100, 10) (6, 10)
```

Tenga en cuenta que la etiqueta, es decir, su y, está incluida en el conjunto de datos. Por lo tanto, debe definir otras dos listas. Uno que

contiene sólo las entidades y otro con el nombre de la etiqueta solamente. Estas dos listas le dirán a su estimador cuáles son las entidades en el dataset y qué nombre de columna es la etiqueta

Se hace con el siguiente código.

```
FEATURES = ["crim", "zn", "indus", "nox", "rm",
            "age", "dis", "tax", "ptratio"]
LABEL = "medv"
```

Paso 2) Convertir los datos

Debe convertir las variables numéricas en el formato adecuado.

Tensorflow proporciona un método para convertir la variable continua: `tf.feature_column.numeric_column()`.

En el paso anterior, se define una lista de una función que desea incluir en el modelo. Ahora puede usar esta lista para convertirlos en datos numéricos. Si desea excluir funciones en el modelo, no dude en soltar una o más variables en la lista FEATURES antes de construir el feature_cols

Tenga en cuenta que utilizará la comprensión de la lista de Python con la lista FEATURES para crear una nueva lista llamada feature_cols. Le ayuda a evitar escribir nueve veces `tf.feature_column.numeric_column()`. Una comprensión de listas es una forma más rápida y limpia de crear nuevas listas

```
feature_cols = [tf.feature_column.numeric_column(k) for k in
                FEATURES]
```

Paso 3) Definir el estimador

En este paso, debe definir el estimador. Tensorflow proporciona actualmente 6 estimadores preconstruidos, incluidos 3 para la tarea de clasificación y 3 para la tarea de regresión:

- Regreso
 - DNNregresor
 - Retrocesor lineal
 - DNNLineaCombinedRegressor
- Clasificador
 - Clasificador DNN
 - Clasificador lineal
 - DNNLineaCombinedClassifier

En este tutorial, utilizará el regresor lineal. Para acceder a esta función, debe usar `tf.estimator`.

La función necesita dos argumentos:

- `feature_columns`: Contiene las variables para incluir en el modelo
- `model_dir`: ruta para almacenar el gráfico, guardar los parámetros del modelo, etc.

Tensorflow creará automáticamente un archivo llamado `train` en su directorio de trabajo. Debe usar esta ruta para acceder al Tensorboard.

```
estimator = tf.estimator.LinearRegressor(
    feature_columns=feature_cols,
    model_dir="train")
```

Salida

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a215dc550>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
```

```
'_num_worker_replicas': 1}
```

La pieza complicada con TensorFlow es la forma de alimentar el modelo. Tensorflow está diseñado para trabajar con computación paralela y conjuntos de datos muy grandes. Debido a la limitación de los recursos de la máquina, es imposible alimentar el modelo con todos los datos a la vez. Para eso, debe alimentar un lote de datos cada vez. Tenga en cuenta que, estamos hablando de un enorme conjunto de datos con millones o más registros. Si no agrega lote, terminará con un error de memoria.

Por ejemplo, si sus datos contienen 100 observaciones y define un tamaño de lote de 10, significa que el modelo verá 10 observaciones para cada iteración ($10 * 10$).

Cuando el modelo ha visto todos los datos, termina una **época**. Una época define cuántas veces desea que el modelo vea los datos. Es mejor establecer este paso en ninguno y dejar que el modelo realiza número de iteración de tiempo.

Una segunda información para agregar es si desea barajar los datos antes de cada iteración. Durante el entrenamiento, es importante mezclar los datos para que el modelo no aprenda un patrón específico del conjunto de datos. Si el modelo aprende los detalles del patrón subyacente de los datos, tendrá dificultades para generalizar la predicción de datos no vistos. Esto se llama **sobreajuste**. El modelo funciona bien en los datos de entrenamiento, pero no puede predecir correctamente los datos no vistos.

TensorFlow hace que estos dos pasos sean fáciles de hacer. Cuando los datos van a la canalización, sabe cuántas observaciones necesita (lote) y si tiene que barajar los datos.

Para instruir a Tensorflow cómo alimentar el modelo, puede usar `pandas_input_fn`. Este objeto necesita 5 parámetros:

- x: datos de entidad
- y: datos de etiqueta
- batch_size: batch. De forma predeterminada 128
- num_epoch: Número de época, por defecto 1
- shuffle: barajar o no los datos. De forma predeterminada, Ninguno

Necesita alimentar el modelo muchas veces para definir una función para repetir este proceso. toda esta función get_input_fn.

```
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

El método habitual para evaluar el rendimiento de un modelo es:

- Entrenar el modelo
- Evaluar el modelo en un dataset diferente
- Predicción

El estimador Tensorflow proporciona tres funciones diferentes para llevar a cabo estos tres pasos fácilmente.

Paso 4): Entrenar el modelo

Puede utilizar el tren del estimador para evaluar el modelo. El estimador del tren necesita un input_fn y una serie de pasos. Puede utilizar la función que creó anteriormente para alimentar el modelo. A continuación, instruye al modelo para iterar 1000 veces. Tenga en cuenta que, si no especifica el número de épocas, deja que el modelo se repita 1000 veces. Si establece el número de época en 1, entonces el modelo iterará 4 veces: Hay 400 registros en el conjunto de entrenamiento, y el

tamaño del lote es 128

1. 128 filas
2. 128 filas
3. 128 filas
4. 16 filas

Por lo tanto, es más fácil establecer el número de época en ninguno y definir el número de iteración.

```
estimator.train(input_fn=get_input_fn(training_set,
                                         num_epochs=None,
                                         n_batch = 128,
                                         shuffle=False),
                                         steps=1000)
```

Salida

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 238.616  
INFO:tensorflow:loss = 13909.657, step = 101 (0.420 sec)  
INFO:tensorflow:global_step/sec: 314.293  
INFO:tensorflow:loss = 12881.449, step = 201 (0.320 sec)  
INFO:tensorflow:global_step/sec: 303.863  
INFO:tensorflow:loss = 12391.541, step = 301 (0.327 sec)  
INFO:tensorflow:global_step/sec: 308.782  
INFO:tensorflow:loss = 12050.5625, step = 401 (0.326 sec)  
INFO:tensorflow:global_step/sec: 244.969  
INFO:tensorflow:loss = 11766.134, step = 501 (0.407 sec)  
INFO:tensorflow:global_step/sec: 155.966  
INFO:tensorflow:loss = 11509.922, step = 601 (0.641 sec)  
INFO:tensorflow:global_step/sec: 263.256  
INFO:tensorflow:loss = 11272.889, step = 701 (0.379 sec)  
INFO:tensorflow:global_step/sec: 254.112
```

```
INFO:tensorflow:loss = 11051.9795, step = 801 (0.396 sec)
INFO:tensorflow:global_step/sec: 292.405
INFO:tensorflow:loss = 10845.855, step = 901 (0.341 sec)
INFO:tensorflow:Saving checkpoints for 1000 into train/model.ckpt.
INFO:tensorflow:Loss for final step: 5925.9873.
```

Usted puede comprobar el Tensorboard será el siguiente comando:

```
activate hello-tf
# For MacOS
tensorboard --logdir=./train
# For Windows
tensorboard --logdir=train
```

Paso 5) Evalúe su modelo

Puede evaluar el ajuste de su modelo en el conjunto de pruebas con el siguiente código:

```
ev = estimator.evaluate(
    input_fn=get_input_fn(test_set,
        num_epochs=1,
        n_batch = 128,
        shuffle=False))
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-13-01:43:13
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-13-01:43:13
INFO:tensorflow:Saving dict for global step 1000: average_loss =
32.15896, global_step = 1000, loss = 3215.896
```

Puede imprimir la pérdida con el siguiente código:

```
loss_score = ev["loss"]
print("Loss: {:.f}".format(loss_score))
```

Salida

```
Loss: 3215.895996
```

El modelo tiene una pérdida de 3215. Puede consultar la estadística de resumen para tener una idea de lo grande que es el error.

```
training_set['medv'].describe()
```

Salida

```
count    400.000000
mean     22.625500
std      9.572593
min      5.000000
25%     16.600000
50%     21.400000
75%     25.025000
max     50.000000
Name: medv, dtype: float64
```

De la estadística resumida anterior, usted sabe que el precio promedio de una casa es de 22 mil, con un precio mínimo de 9 miles y máximo de 50 mil. El modelo hace un error típico de 3k dólares.

Paso 6) Hacer la predicción

Finalmente, puede utilizar el estimador predictor para estimar el valor de 6 casas de Boston.

```
y = estimator.predict(
    input_fn=get_input_fn(prediction_set,
    num_epochs=1,
    n_batch = 128,
    shuffle=False))
```

Para imprimir los valores estimados de, puede utilizar este código:

```
predictions = list(p["predictions"] for p in itertools.islice(y,
6))print("Predictions: {}".format(str(predictions)))
```

Salida

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
Predictions: [array([32.297546], dtype=float32), array([18.96125],  
dtype=float32), array([27.270979], dtype=float32),  
array([29.299236], dtype=float32), array([16.436684],  
dtype=float32), array([21.460876], dtype=float32)]
```

El modelo pronostica los siguientes valores:

Casa	Predictión
1	32.29
2	18.96
3	27.27
4	29.29
5	16.43
7	21.46

Tenga en cuenta que no sabemos el verdadero valor de. En el tutorial de aprendizaje profundo, tratarás de vencer al modelo lineal

Solución Numpy

En esta sección se explica cómo entrenar el modelo utilizando un estimador numpy para alimentar los datos. El método es el mismo excepto que utilizará numpy_input_fn estimador.

```
training_set_n = pd.read_csv("E:/boston_train.csv").values
```

```
test_set_n = pd.read_csv("E:/boston_test.csv").values
```

```
prediction_set_n = pd.read_csv("E:/boston_predict.csv").values
```

Paso 1) Importar los datos

En primer lugar, debe diferenciar las variables de entidad de la etiqueta. Debe hacer esto para los datos de capacitación y la evaluación. Es más rápido definir una función para dividir los datos.

```
def prepare_data(df):
    X_train = df[:, :-3]
    y_train = df[:, -3]
    return X_train, y_train
```

Puede usar la función para dividir la etiqueta de las características del conjunto de datos de tren/evaluar

```
X_train, y_train = prepare_data(training_set_n)
X_test, y_test = prepare_data(test_set_n)
```

Debe excluir la última columna del dataset de predicción porque solo contiene NaN

```
x_predict = prediction_set_n[:, :-2]
```

Confirme la forma de la matriz. Tenga en cuenta que, la etiqueta no debe

tener una dimensión, significa (400,).

```
print(X_train.shape, y_train.shape, x_predict.shape)
```

Salida

```
(400, 9) (400,) (6, 9)
```

Puede construir las columnas de entidades de la siguiente manera:

```
feature_columns = [tf.feature_column.numeric_column('x',  
shape=X_train.shape[1:])]
```

El estimador se define como antes, se indica a las columnas de entidad y dónde guardar el gráfico.

```
estimator = tf.estimator.LinearRegressor(  
    feature_columns=feature_columns,  
    model_dir="train1")
```

Salida

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'train1',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,  
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':  
100, '_train_distribute': None, '_service': None, '_cluster_spec':  
<tensorflow.python.training.server_lib.ClusterSpec object at  
0x1a218d8f28>, '_task_type': 'worker', '_task_id': 0,  
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':  
'', '_is_chief': True, '_num_ps_replicas': 0,  
'_num_worker_replicas': 1}
```

Puede usar el estimador numpy para alimentar los datos al modelo y luego entrenar el modelo. Tenga en cuenta que, definimos la función input_fn antes para facilitar la legibilidad.

```
# Train the estimator  
train_input =
```

```
tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train,
    batch_size=128,
    shuffle=False,
    num_epochs=None)
estimator.train(input_fn = train_input, steps=5000)
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into train1/model.ckpt.
INFO:tensorflow:loss = 83729.64, step = 1
INFO:tensorflow:global_step/sec: 490.057
INFO:tensorflow:loss = 13909.656, step = 101 (0.206 sec)
INFO:tensorflow:global_step/sec: 788.986
INFO:tensorflow:loss = 12881.45, step = 201 (0.126 sec)
INFO:tensorflow:global_step/sec: 736.339
INFO:tensorflow:loss = 12391.541, step = 301 (0.136 sec)
INFO:tensorflow:global_step/sec: 383.305
INFO:tensorflow:loss = 12050.561, step = 401 (0.260 sec)
INFO:tensorflow:global_step/sec: 859.832
INFO:tensorflow:loss = 11766.133, step = 501 (0.117 sec)
INFO:tensorflow:global_step/sec: 804.394
INFO:tensorflow:loss = 11509.918, step = 601 (0.125 sec)
INFO:tensorflow:global_step/sec: 753.059
INFO:tensorflow:loss = 11272.891, step = 701 (0.134 sec)
INFO:tensorflow:global_step/sec: 402.165
INFO:tensorflow:loss = 11051.979, step = 801 (0.248 sec)
INFO:tensorflow:global_step/sec: 344.022
INFO:tensorflow:loss = 10845.854, step = 901 (0.288 sec)
INFO:tensorflow:Saving checkpoints for 1000 into train1/model.ckpt.
INFO:tensorflow:Loss for final step: 5925.985.
Out[23]:
<tensorflow.python.estimator.canned.linear.LinearRegressor at
0x1a1b6ea860>
```

Replica el mismo paso con un estimador diferente para evaluar el modelo

```
eval_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_test},
    y=y_test,
    shuffle=False,
    batch_size=128,
    num_epochs=1)
estimator.evaluate(eval_input, steps=None)
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-13-01:44:00
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-13-01:44:00
INFO:tensorflow:Saving dict for global step 1000: average_loss =
32.158947, global_step = 1000, loss = 3215.8945
Out[24]:
{'average_loss': 32.158947, 'global_step': 1000, 'loss': 3215.8945}
```

Finalmente, puede calcular la predicción, debe ser similar a los pandas.

```
test_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": x_predict},
    batch_size=128,
    num_epochs=1,
    shuffle=False)
y = estimator.predict(test_input)
predictions = list(p["predictions"] for p in itertools.islice(y, 6))
print("Predictions: {}".format(str(predictions)))
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
```

```
Predictions: [array([32.297546], dtype=float32), array([18.961248],  
dtype=float32), array([27.270979], dtype=float32),  
array([29.299242], dtype=float32), array([16.43668],  
dtype=float32), array([21.460878], dtype=float32)]
```

Solución Tensorflow

La última sección está dedicada a una solución TensorFlow. Este método es poco más complicado que el otro.

Tenga en cuenta que si utiliza Jupyter notebook, necesita reiniciar y limpiar el núcleo para ejecutar esta sesión.

TensorFlow ha construido una gran herramienta para pasar los datos a la tubería. En esta sección, construirá usted mismo la función `input_fn`.

Paso 1) Definir la ruta y el formato de los datos

En primer lugar, declara dos variables con la ruta del archivo csv. Tenga en cuenta que tiene dos archivos, uno para el conjunto de entrenamiento y otro para el conjunto de pruebas.

```
import tensorflow as tf  
  
df_train = "E:/boston_train.csv"  
  
df_eval = "E:/boston_test.csv"
```

Luego, debe definir las columnas que desea usar desde el archivo csv. Usaremos todo. Después de eso, debe declarar el tipo de variable que es.

La variable Floats se define por [0.]

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",  
           "dis", "tax", "ptratio", "medv"]RECORDS_ALL =  
[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]]
```

Paso 2) Definir la función `input_fn`

La función se puede dividir en tres partes:

1. Importar los datos
2. Crear el iterador
3. Consumir los datos

A continuación se muestra el código general para definir la función. el código se explicará después de

```
def input_fn(data_file, batch_size, num_epoch = None):
    # Step 1
    def parse_csv(value):
        columns = tf.decode_csv(value, record_defaults=
RECORDS_ALL)
        features = dict(zip(COLUMNS, columns))
        #labels = features.pop('median_house_value')
        labels = features.pop('medv')
        return features, labels

    # Extract lines from input files using the
    # Dataset API.
    dataset =
(tf.data.TextLineDataset(data_file) # Read text file
 .skip(1) # Skip header row
 .map(parse_csv))

    dataset = dataset.repeat(num_epoch)
    dataset = dataset.batch(batch_size)
    # Step 3
    iterator = dataset.make_one_shot_iterator()
    features, labels = iterator.get_next()
    return features, labels
```

**** Importar los datos ****

Para un archivo csv, el método de dataset lee una línea a la vez. Para crear el dataset, debe utilizar el objeto TextLineDataset. Su dataset tiene un encabezado, por lo que necesita usar skip (1) para omitir la primera línea. En este punto, solo lee los datos y excluye el encabezado en la canalización. Para alimentar el modelo, debe separar las funciones de la etiqueta. El método utilizado para aplicar cualquier transformación a los datos es el mapa.

Este método llama a una función que va a crear para instruir cómo transformar los datos. En pocas palabras, debe pasar los datos en el objeto TextLineDataset, excluir el encabezado y aplicar una transformación que se indica por una explicación Function.Code

- tf.data.textlineDataset (data_file): Esta línea lee el archivo csv
- .skip (1): omite el encabezado
- .map (parse_csv)): analizar los registros en los tensors. Debe definir una función para instruir al objeto de mapa. Puede llamar a esta función parse_csv.

Esta función analiza el archivo csv con el método tf.decode_csv y declara las características y la etiqueta. Las entidades se pueden declarar como un diccionario o una tupla. Utiliza el método de diccionario porque es más convenient. Code explicación

- tf.decode_csv (value, record_defaults = RECORDS_ALL): el método decode_csv utiliza la salida del TextLineDataset para leer el archivo csv. record_defaults indica a TensorFlow sobre el tipo de columnas.
- dict (zip (_CSV_COLUMNS, columns)): Rellene el diccionario con todas las columnas extraídas durante este procesamiento de datos
- features.pop ('median_house_value'): Excluir la variable de destino de la variable de entidad y crear una variable de etiqueta

El conjunto de datos necesita más elementos para alimentar iterativamente los tensores. De hecho, debe agregar la repetición del método para permitir que el conjunto de datos continúe indefinidamente para alimentar el modelo. Si no agrega el método, el modelo iterará solo una vez y luego arrojará un error porque no se ingresarán más datos en la canalización.

Después de eso, puede controlar el tamaño del lote con el método por lotes. Significa que le dice al conjunto de datos cuántos datos desea pasar

en la canalización para cada iteración. Si establece un tamaño de lote grande, el modelo será lento.

Paso 3) Crear el iterador

Ahora está listo para el segundo paso: crear un iterador para devolver los elementos en el dataset.

La forma más sencilla de crear un operador es con el método make_one_shot_iterator.

Después de eso, puede crear las entidades y etiquetas desde el iterador.

Paso 4) Consumir los datos

Puede comprobar lo que sucede con la función input_fn. Debe llamar a la función en una sesión para consumir los datos. Intenta con un tamaño de lote igual a 1.

Tenga en cuenta que, imprime las entidades en un diccionario y la etiqueta como una matriz.

Se mostrará la primera línea del archivo csv. Puede intentar ejecutar este código muchas veces con diferentes tamaños de lote.

```
next_batch = input_fn(df_train, batch_size = 1, num_epoch = None)
with tf.Session() as sess:
    first_batch = sess.run(next_batch)
    print(first_batch)
```

Salida

```
({'crim': array([2.3004], dtype=float32), 'zn': array([0.],
dtype=float32), 'indus': array([19.58], dtype=float32), 'nox':
array([0.605], dtype=float32), 'rm': array([6.319], dtype=float32),
'age': array([96.1], dtype=float32), 'dis': array([2.1],
dtype=float32), 'tax': array([403.], dtype=float32), 'ptratio':
```

```
array([14.7], dtype=float32)}, array([23.8], dtype=float32))
```

Paso 4) Definir la columna de entidad

Debe definir las columnas numéricas de la siguiente manera:

```
X1= tf.feature_column.numeric_column('crim')
X2= tf.feature_column.numeric_column('zn')
X3= tf.feature_column.numeric_column('indus')
X4= tf.feature_column.numeric_column('nox')
X5= tf.feature_column.numeric_column('rm')
X6= tf.feature_column.numeric_column('age')
X7= tf.feature_column.numeric_column('dis')
X8= tf.feature_column.numeric_column('tax')
X9= tf.feature_column.numeric_column('ptratio')
```

Tenga en cuenta que necesita combinar todas las variables en un cubo

```
base_columns = [X1, X2, X3,X4, X5, X6,X7, X8, X9]
```

Paso 5) Construir el modelo

Puede entrenar el modelo con el estimador LineArregresor.

```
model = tf.estimator.LinearRegressor(feature_columns=base_columns,
model_dir='train3')
```

Salida

```
INFO:tensorflow:Using default config. INFO:tensorflow:Using config:
{'_model_dir': 'train3', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None,
'_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1820a010f0>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

Necesita usar una función lambda para permitir escribir el argumento en la función inpu_fn. Si no utiliza una función lambda, no puede entrenar el modelo.

```
# Train the estimator
model.train(steps =1000,
            input_fn= lambda : input_fn(df_train,batch_size=128,
num_epoch = None))
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into train3/model.ckpt.
INFO:tensorflow:loss = 83729.64, step = 1
INFO:tensorflow:global_step/sec: 72.5646
INFO:tensorflow:loss = 13909.657, step = 101 (1.380 sec)
INFO:tensorflow:global_step/sec: 101.355
INFO:tensorflow:loss = 12881.449, step = 201 (0.986 sec)
INFO:tensorflow:global_step/sec: 109.293
INFO:tensorflow:loss = 12391.541, step = 301 (0.915 sec)
INFO:tensorflow:global_step/sec: 102.235
INFO:tensorflow:loss = 12050.5625, step = 401 (0.978 sec)
INFO:tensorflow:global_step/sec: 104.656
INFO:tensorflow:loss = 11766.134, step = 501 (0.956 sec)
INFO:tensorflow:global_step/sec: 106.697
INFO:tensorflow:loss = 11509.922, step = 601 (0.938 sec)
INFO:tensorflow:global_step/sec: 118.454
INFO:tensorflow:loss = 11272.889, step = 701 (0.844 sec)
INFO:tensorflow:global_step/sec: 114.947
INFO:tensorflow:loss = 11051.9795, step = 801 (0.870 sec)
INFO:tensorflow:global_step/sec: 111.484
INFO:tensorflow:loss = 10845.855, step = 901 (0.897 sec)
INFO:tensorflow:Saving checkpoints for 1000 into train3/model.ckpt.
INFO:tensorflow:Loss for final step: 5925.9873.
Out[8]:
<tensorflow.python.estimator.canned.linear.LinearRegressor at
0x18225eb8d0>
```

Puede evaluar el ajuste de su modelo en el conjunto de pruebas con el siguiente código:

```
results = model.evaluate(steps =None, input_fn=lambda:  
    input_fn(df_eval, batch_size =128, num_epoch = 1))  
for key in results:  
    print("    {}, was: {}".format(key, results[key]))
```

Salida

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-13-02:06:02  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-13-02:06:02  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
32.15896, global_step = 1000, loss = 3215.896  
    average_loss, was: 32.158958435058594  
    loss, was: 3215.89599609375  
    global_step, was: 1000
```

El último paso es predecir el valor de basado en el valor de, las matrices de las entidades. Puede escribir un diccionario con los valores que desea predecir. El modelo tiene 9 funciones, por lo que debe proporcionar un valor para cada una de ellas. El modelo proporcionará una predicción para cada uno de ellos.

En el siguiente código, escribió los valores de cada una de las entidades contenidas en el archivo csv df_predict.

Debe escribir una nueva función input_fn porque no hay etiqueta en el dataset. Puede usar la API from_tensor del conjunto de datos.

```
prediction_input = {  
    'crim':  
[0.03359, 5.09017, 0.12650, 0.05515, 8.15174, 0.24522],
```

```

'zn': [75.0,0.0,25.0,33.0,0.0,0.0],
'indus': [2.95,18.10,5.13,2.18,18.10,9.90],
'nox': [0.428,0.713,0.453,0.472,0.700,0.544],
'rm': [7.024,6.297,6.762,7.236,5.390,5.782],
'age': [15.8,91.8,43.4,41.1,98.9,71.7],
'dis': [5.4011,2.3682,7.9809,4.0220,1.7281,4.0317],
'tax': [252,666,284,222,666,304],
'ptratio': [18.3,20.2,19.7,18.4,20.2,18.4]
}
def test_input_fn():
dataset = tf.data.Dataset.from_tensors(prediction_input)
return dataset

# Predict all our prediction_input
pred_results =
model.predict(input_fn=test_input_fn)

```

Finalmente, imprime las predicciones.

```

for pred in enumerate(pred_results):
print(pred)

```

Salida

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
(0, {'predictions': array([32.297546], dtype=float32)})
(1, {'predictions': array([18.96125], dtype=float32)})
(2, {'predictions': array([27.270979], dtype=float32)})
(3, {'predictions': array([29.299236], dtype=float32)})
(4, {'predictions': array([16.436684], dtype=float32)})
(5, {'predictions': array([21.460876], dtype=float32)})

INFO:tensorflow:Calling model_fn. INFO:tensorflow:Done calling
model_fn. INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train3/model.ckpt-5000
INFO:tensorflow:Running local_init_op. INFO:tensorflow:Done running
local_init_op. (0, {'predictions': array([35.60663],
dtype=float32)}) (1, {'predictions': array([22.298521],
dtype=float32)}) (2, {'predictions': array([25.74533],
dtype=float32)}) (3, {'predictions': array([35.126694],

```

```
dtype=float32)}) (4, {'predictions': array([17.94416],  
dtype=float32)}) (5, {'predictions': array([22.606628],  
dtype=float32)})
```

Resumen

Para entrenar un modelo, debe:

- Definir las funciones: Variables independientes: X
- Definir la etiqueta: Variable dependiente: y
- Construir un conjunto de tren/prueba
- Definir el peso inicial
- Definir la función de pérdida: MSE
- Optimizar el modelo: Descenso degradado
- Definir:
 - Tasa de aprendizaje
 - Número de época
 - Tamaño del lote

En este tutorial, aprendió a utilizar la API de alto nivel para un estimador de regresión lineal. Debe definir:

1. Columnas de entidades. Si es continuo:
`tf.feature_column.numeric_column()`. Puede llenar una lista con comprensión de lista de Python
2. El estimador: `tf.estimator.linearRegressor(feature_columns, model_dir)`
3. Una función para importar los datos, el tamaño del lote y la época:
`input_fn()`

Después de eso, usted está listo para entrenar, evaluar y hacer predicciones con `train()`, `evaluate()` y `predict()`

Capítulo 14: Estudio de caso de regresión lineal

En este tutorial, aprenderá a comprobar los datos y prepararlos para crear una tarea de regresión lineal.

Este tutorial se divide en dos partes:

- Buscar interacción
- Probar el modelo

En el tutorial anterior, utilizó el dataset de Boston para estimar el precio medio de una casa. El conjunto de datos de Boston tiene un tamaño pequeño, con solo 506 observaciones. Este conjunto de datos se considera como un punto de referencia para probar nuevos algoritmos de regresión lineal.

El conjunto de datos se compone de:

Variable	Descripción
zn	Proporción de terrenos residenciales zonificados para lotes de más de 25.000 pies cuadrados.
indus	La proporción de acres de empresas no minoristas por ciudad.
nox	concentración de óxidos nítricos
rm	promedio de habitaciones por vivienda
edad	la proporción de unidades ocupadas por los propietarios construidas antes de 1940
dis	distancias ponderadas a cinco centros de empleo de Boston
impuestos	valor total de la propiedad de la tasa de impuesto por dólar 10.000
ptratio	la proporción alumnos/maestro por una ciudad
medv	El valor medio de las viviendas ocupadas por los propietarios en miles de dólares

crim	Tasa de delincuencia per cápita por ciudad
chas	Variable ficticia de Charles River (1 si limita el río; 0 en caso contrario)
B	la proporción de población negra por la ciudad

En este tutorial, vamos a estimar el precio medio usando un regresor lineal, pero el foco está en un proceso particular de aprendizaje automático: “preparación de datos”.

Un modelo generaliza el patrón en los datos. Para capturar tal patrón, primero debe encontrarlo. Una buena práctica es realizar un análisis de datos antes de ejecutar cualquier algoritmo de aprendizaje automático.

Elegir las funciones correctas marca la diferencia en el éxito de su modelo. Imagina que intentas estimar el salario de un pueblo, si no inclues el género como una covariada, terminas con una estimación pobre.

Otra forma de mejorar el modelo es mirar la correlación entre la variable independiente. Volviendo al ejemplo, se puede pensar en la educación como un excelente candidato para predecir el salario, pero también la ocupación. Es justo decir que la ocupación depende del nivel de educación, es decir, la educación superior a menudo conduce a una mejor ocupación. Si generalizamos esta idea, podemos decir que la correlación entre la variable dependiente y una variable explicativa puede ampliarse de otra variable explicativa.

Para captar el efecto limitado de la educación sobre la ocupación, podemos utilizar un término de interacción.

INTERACTION TERM

Interaction terms allow us model relationships when the effects of a feature on the target is influenced by another feature.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + e$$

The interaction of features
 x_1 and x_2 .

ChrisAlbon

Si nos fijamos en la ecuación salarial, se convierte en:

$$\text{wage} = \alpha + \beta_1 \text{occupation} + \beta_2 \text{education} + \beta_3 \text{occupation} * \text{education} + \epsilon$$

Si β_3 es positivo, entonces implica que un nivel adicional de educación produce un aumento mayor en el valor medio de una casa para un alto nivel de ocupación. En otras palabras, existe un efecto de interacción entre la educación y la ocupación.

En este tutorial, vamos a tratar de ver qué variables pueden ser un buen candidato para términos de interacción. Vamos a probar si la adición de este tipo de información conduce a una mejor predicción de precios.

Resumen de estadísticas

Hay algunos pasos que puede seguir antes de continuar con el modelo. Como se mencionó anteriormente, el modelo es una generalización de los datos. La mejor práctica es entender los datos y hacer una predicción. Si no conoce sus datos, tiene pocas posibilidades de mejorar su modelo.

Como primer paso, cargue los datos como un marco de datos pandas y cree un conjunto de entrenamiento y un conjunto de pruebas.

Consejos: Para este tutorial, necesita tener matplotlib y seaborn instalados en Python. Puede instalar el paquete Python sobre la marcha con Jupyter. A ti **No debería** hacer esto

```
!conda install --yes matplotlib
```

pero

```
import sys
!{sys.executable} -m pip install matplotlib # Already installed
!{sys.executable} -m pip install seaborn
```

Tenga en cuenta que este paso no es necesario si tiene matplotlib y seaborn instalados.

Matplotlib es la biblioteca para crear un gráfico en Python. Seaborn es una biblioteca de visualización estadística construida sobre matplotlib. Proporciona parcelas atractivas y hermosas.

El siguiente código importa las bibliotecas necesarias.

```
import pandas as pd
from sklearn import datasets
import tensorflow as tf
from sklearn.datasets import load_boston
```

```
import numpy as np
```

La biblioteca sklearn incluye el conjunto de datos de Boston. Puede llamar a su API para importar los datos.

```
boston = load_boston()  
df = pd.DataFrame(boston.data)
```

El nombre de la entidad se almacena en el objeto feature_names en una matriz.

```
boston.feature_names
```

Salida

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',  
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='|<U7')
```

Puede cambiar el nombre de las columnas.

```
df.columns = boston.feature_names  
df['PRICE'] = boston.target  
df.head(2)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6

Convertir la variable CHAS como una variable de cadena y etiquetarla con yes si CHAS = 1 y no si CHAS = 0

```
df['CHAS'] = df['CHAS'].map({1:'yes', 0:'no'})  
df['CHAS'].head(5)  
0    no  
1    no  
2    no  
3    no  
4    no  
Name: CHAS, dtype: object
```

Con pandas, es sencillo dividir el conjunto de datos. Divide aleatoriamente el conjunto de datos con un conjunto de entrenamiento del 80 por ciento y un conjunto de pruebas del 20 por ciento. Los pandas tienen una función incorporada para dividir una muestra de marco de datos.

El primer parámetro `frac` es un valor de 0 a 1. Se establece en 0,8 para seleccionar aleatoriamente 80 por ciento del marco de datos.

`Random_state` permite tener el mismo marco de datos devuelto para todos.

```
### Create train/test set
df_train=df.sample(frac=0.8,random_state=200)
df_test=df.drop(df_train.index)
```

Puede obtener la forma de los datos. Debe ser:

- Set de tren: $506 \times 0.8 = 405$
- Juego de pruebas: $506 \times 0.2 = 101$

```
print(df_train.shape, df_test.shape)
```

Salida

```
(405, 14) (101, 14)
```

```
df_test.head(5)
```

Salida

	CRIM	ZN	INDUS	CHAS	NOX	RM	EDAD	DIS	RAD	IMPUESTOS	I
0	0.00632	18.0	2.31	no	0.538	6.575	65.2	4.0900	1.0	296.0	1
1	0.02731	0.0	7.07	no	0.469	6.421	78.9	4.9671	2.0	242.0	1
3	0.03237	0.0	2.18	no	0.458	6.998	45.8	6.0622	3.0	222.0	1
6	0.08829	12.5	7.87	no	0.524	6.012	66.6	5.5605	5.0	311.0	1
7	0.14455	12.5	7.87	no	0.524	6.172	96.1	5.9505	5.0	311.0	1

Los datos son desordenados; a menudo están mal equilibrados y salpicados de valores atípicos que descartan el entrenamiento de análisis y aprendizaje automático.

El primer paso para limpiar el conjunto de datos es comprender dónde necesita limpieza. La limpieza de un conjunto de datos puede ser difícil de hacer, especialmente en cualquier forma generalizable

El equipo de Google Research ha desarrollado una herramienta para este trabajo llamada **Facetas** que ayudan a visualizar los datos y cortarlos de todo tipo de maneras. Este es un buen punto de partida para comprender cómo se presenta el conjunto de datos.

Las facetas le permiten encontrar dónde los datos no se ven del todo de la forma en que está pensando.

A excepción de su aplicación web, Google facilita la inserción del kit de herramientas en un cuaderno de Jupyter.

Hay dos partes en las facetas:

- Descripción general de las facetas
- Inmersión profunda de facetas

Descripción general de las facetas

Descripción general de las facetas proporciona una visión general del conjunto de datos. Descripción general de facetas divide las columnas de los datos en filas de información sobresaliente que muestra

1. el porcentaje de observación faltante
2. valores mín y máx.
3. estadísticas como la media, la mediana y la desviación estándar.
4. También agrega una columna que muestra el porcentaje de valores que son ceros, lo que resulta útil cuando la mayoría de los valores son ceros.
5. Es posible ver estas distribuciones en el conjunto de datos de prueba, así como el conjunto de entrenamiento para cada entidad. Esto significa que puede verificar dos veces que la prueba tiene una distribución similar a la del conjunto de datos de capacitación.

Esto es al menos el mínimo que debe hacer antes de cualquier tarea de aprendizaje automático. Con esta herramienta, no te pierdas este paso crucial, y destaca algunas anomalías.

Inmersión profunda de facetas

Facets Deep Dive es una herramienta genial. Permite tener cierta claridad en su conjunto de datos y ampliar todo el camino para ver una pieza individual de datos. Esto significa que puede facetar los datos por fila y columna en cualquiera de las entidades del dataset.

Usaremos estas dos herramientas con el conjunto de datos de Boston.

Nota: No se puede utilizar Vista general de facetas ni Dive profundo de facetas al mismo tiempo. Primero debe borrar el bloc de notas para cambiar la herramienta.

Instalar faceta

Puede utilizar la aplicación web Facet para la mayor parte del análisis. En este tutorial, verá cómo usarlo dentro de un cuaderno Jupyter.

En primer lugar, debe instalar nbextensions. Se hace con este código. Copie y pegue el siguiente código en el terminal de su máquina.

```
pip install jupyter_contrib_nbextensions
```

Inmediatamente después de eso, debe clonar los repositorios en su computadora. Tiene dos opciones:

Opción 1) Copie y pegue este código en el terminal (**Recomendado**)

Si no tiene Git instalado en su máquina, vaya a esta URL <https://git-scm.com/download/win> y siga las instrucciones. Una vez que haya terminado, puede usar el comando git en el terminal para Mac User o Anaconda prompt para el usuario de Windows

```
git clone https://github.com/PAIR-code/facets
```

Opción 2) Ir a <https://github.com/PAIR-code/facets> y descargue los repositorios.

The screenshot shows a GitHub repository page for 'PAIR-code/facets'. At the top, it displays statistics: 105 commits, 16 branches, 2 releases, 17 contributors, and Apache-2.0 license. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and a prominent green 'Clone or download' button. A list of commits is shown, with the most recent being a merge pull request from 'PAIR-code/searchstring'. On the right side, there are download options: 'Clone with HTTPS' (with a link to <https://github.com/PAIR-code/facets.git>) and 'Use SSH'. Below these are 'Open in Desktop' and 'Download ZIP' buttons. A red callout bubble points to the 'Download ZIP' button with the text 'Haga clic aquí para descargar Facets'.

Commit	Message	Date
jameswex Merge pull request #131 from PAIR-code/searchstring	Merge pull request #131 from PAIR-code/searchstring	11 months ago
facets-dist	added rank histogram to custom stats	11 months ago
facets	fix facets jupyter build and distribute new facets dist compi	11 months ago
facets_dive	fix bug with selected indices	11 months ago
facets_overview	expose search string publically	11 months ago
img	adding images for readme	11 months ago
.gitignore	Initial commit	11 months ago
AUTHORS	Initial commit	11 months ago
CONTRIBUTING.md	Initial commit	11 months ago
CONTRIBUTORS	Initial commit	11 months ago
LICENSE	Initial commit	11 months ago
README.md	Add blank lines before lists	9 months ago
WORKSPACE	Add compile=True to HTML binary rules	2 months ago

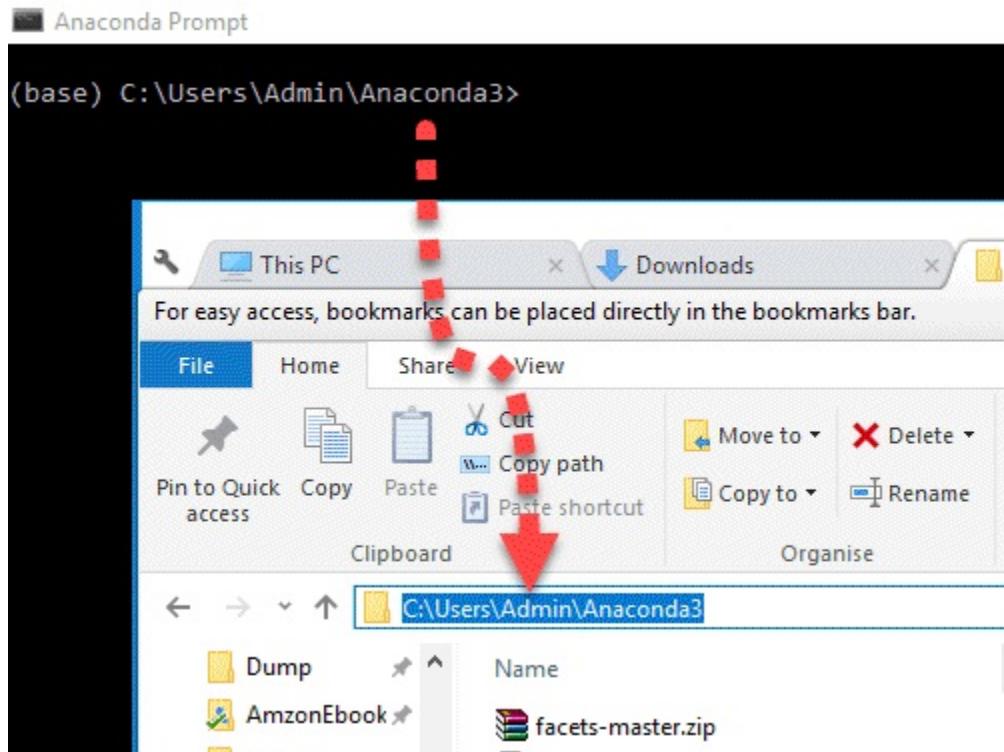
Si elige la primera opción, el archivo termina en el archivo de descarga. Puede dejar que el archivo se descargue o arrastrarlo a otra ruta.

Puede comprobar dónde se almacenan las facetas con esta línea de comandos:

```
echo `pwd`/`ls facets`
```

Ahora que ha localizado Facets, debe instalarlo en Jupyter Notebook. Debe establecer el directorio de trabajo en la ruta donde se encuentran las facetas.

Su directorio de trabajo actual y la ubicación de las facetas zip deben ser los mismos.



Debe apuntar el directorio de trabajo a Facet:

```
cd facets
```

Para instalar Facetas en Jupyter, tiene dos opciones. Si instaló Jupyter con Conda para todos los usuarios, copie este código:

puede usar jupyter nbextension install facets-dist/

```
jupyter nbextension install facets-dist/
```

De lo contrario, utilice:

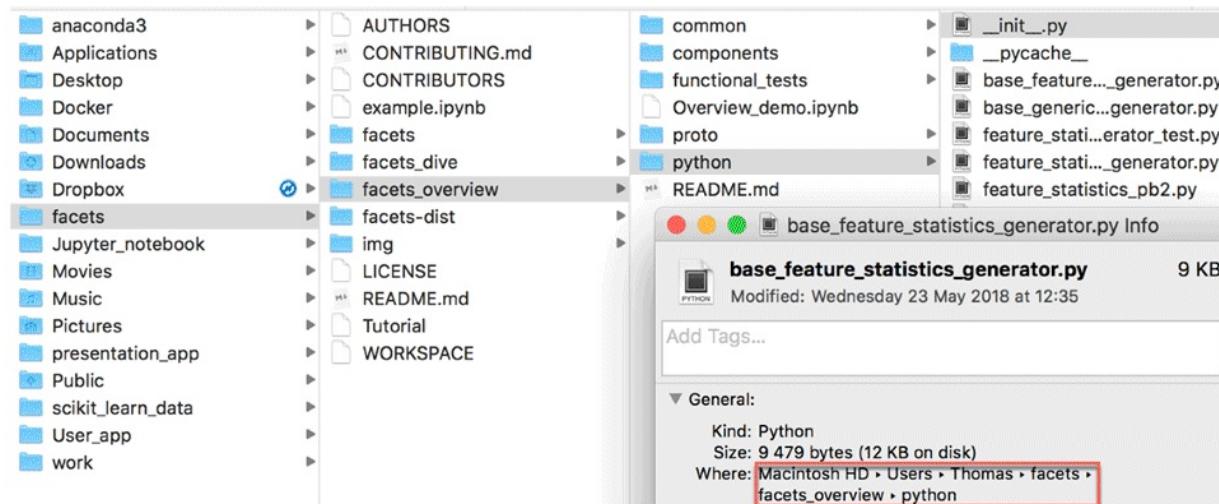
```
jupyter nbextension install facets-dist/ --user
```

Muy bien, ya está listo. Vamos a abrir Facet Overview.

Resumen

Descripción general utiliza una secuencia de comandos de Python para calcular las estadísticas. Debe importar el script denominado generic_feature_statistics_generator a Jupyter. No se preocupe; el script se encuentra en los archivos de facetas.

Necesitas localizar su camino. Es fácil de hacer. Abra facetas, abra el archivo facets_overview y luego python. Copiar la ruta



Después de eso, vuelva a Jupyter y escriba el siguiente código '/Users/Thomas/facets/facets_overview/python' to your path.

```
# Add the facets overview python code to the python path# Add t
import sys
sys.path.append('/Users/Thomas/facets/facets_overview/python')
```

Puede importar el script con el siguiente código.

```
from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

En Windows, el mismo código se convierte en

```
import sys
sys.path.append(r"C:\Users\Admin\Anaconda3\facets-
master\facets_overview\python")

from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

Para calcular las estadísticas de entidad, debe utilizar la función GenericFeatureStatisticsGenerator () y utilizar el objeto ProtoFromDataFrames. Puede pasar el marco de datos en un diccionario. Por ejemplo, si queremos crear una estadística de resumen para el conjunto de trenes, podemos almacenar la información en un diccionario y usarla en el objeto `ProtoFromDataFrames`

- 'name': 'train', 'table': df_train

Nombre es el nombre de la tabla que se muestra y se utiliza el nombre de la tabla que desea calcular el resumen. En su ejemplo, la tabla que contiene los datos es df_train

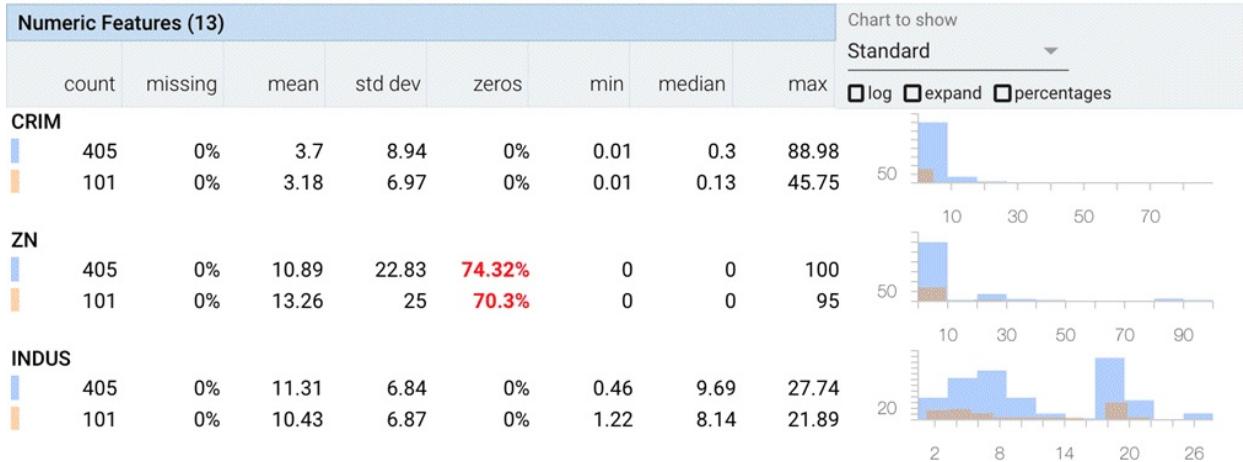
```
# Calculate the feature statistics proto from the datasets and
stringify it for use in facets overview
import base64

gfsg = GenericFeatureStatisticsGenerator()

proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train},
                                  {'name': 'test', 'table':
df_test}])

#proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train}])
protostr = base64.b64encode(proto.SerializeToString()).decode("utf-
8")
```

Por último, simplemente copia y pega el código a continuación. El código viene directamente de GitHub. Deberías poder ver esto:



```
# Display the facets overview visualization for this data# Disp
from IPython.core.display import display, HTML

HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-
dist/facets-jupyter.html" >
    <facets-overview id="elem"></facets-overview>
    <script>
        document.querySelector("#elem").protoInput =
{protostr}";</script>"""
html = HTML_TEMPLATE.format(protostr=protostr)
display(HTML(html))
```

Gráfico

Después de comprobar los datos y su distribución, puede trazar una matriz de correlación. La matriz de correlación calcula el coeficiente de Pearson. Este coeficiente se une entre -1 y 1, con un valor positivo indica una correlación positiva y un valor negativo una correlación negativa.

Está interesado en ver qué variables pueden ser un buen candidato para términos de interacción.

```
## Choose important feature and further check with Dive
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks")
# Compute the correlation matrix
corr = df.corr('pearson')
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

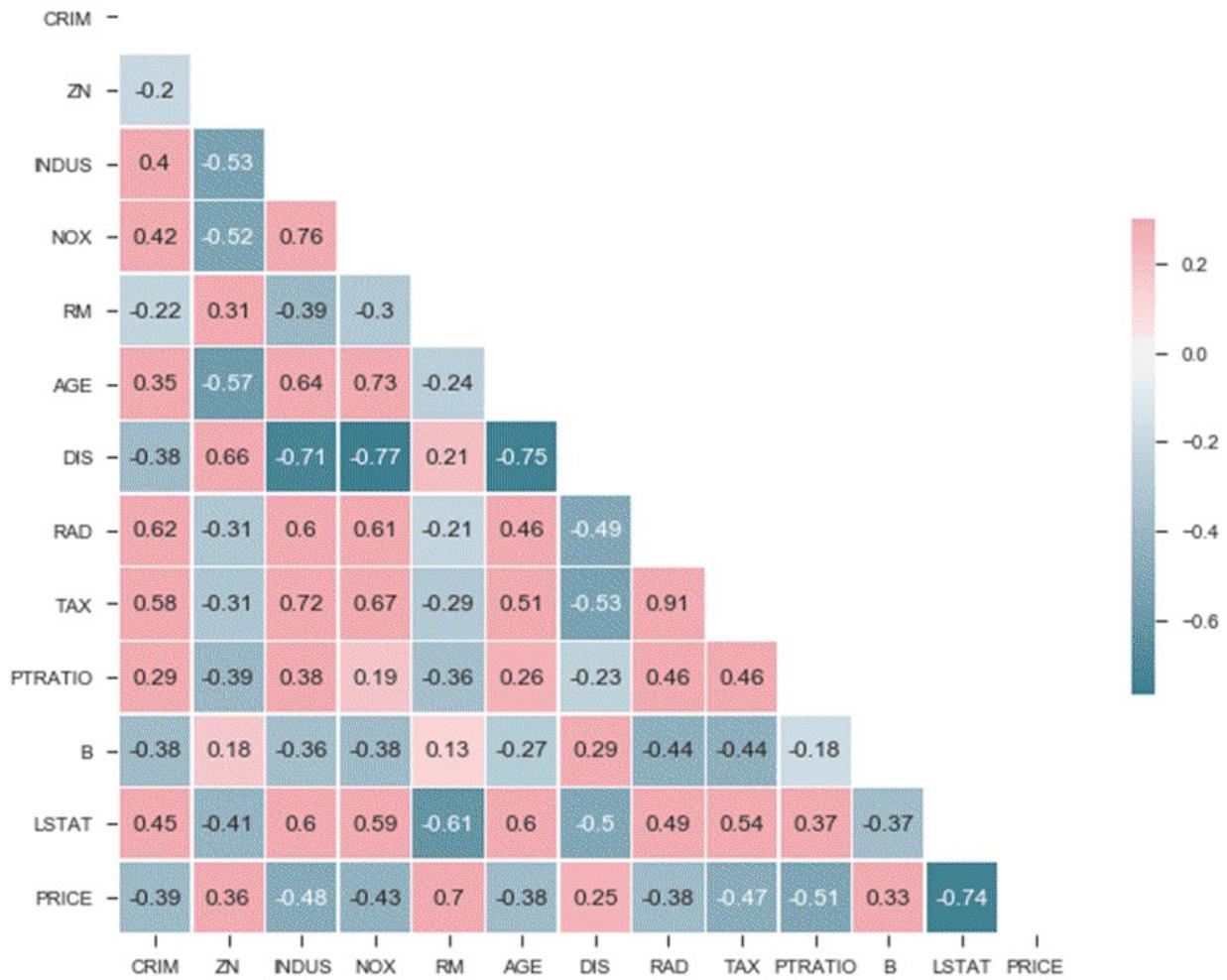
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            center=0, annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Salida

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a184d6518>
```

png



Desde la matriz, se puede ver:

- LSTAT
- RM

Están fuertemente correlacionados con PRECIO. Otra característica interesante es la fuerte correlación positiva entre NOX e INDUS, lo que significa que esas dos variables se mueven en la misma dirección. Además, también se correlacionan con el PRECIO. El DIS también está muy relacionado con el IND y el NOX.

Usted tiene una primera pista de que IND y NOX pueden ser buenos

candidatos para el término de interacción y DIS también podría ser interesante para enfocarse.

Puede ir un poco más profundo trazando una cuadrícula de par. Ilustrará con más detalle el mapa de correlación que trazó antes.

La rejilla par que estamos compuestos de la siguiente manera:

- Parte superior: Parcela de dispersión con línea ajustada
- Diagonal: Gráfico de densidad del núcleo
- Parte inferior: gráfico de densidad del núcleo multivariado

Usted elige el enfoque en cuatro variables independientes. La elección corresponde a las variables con fuerte correlación con PRECIO

- INDUS
- NOX
- RM
- LSTAT

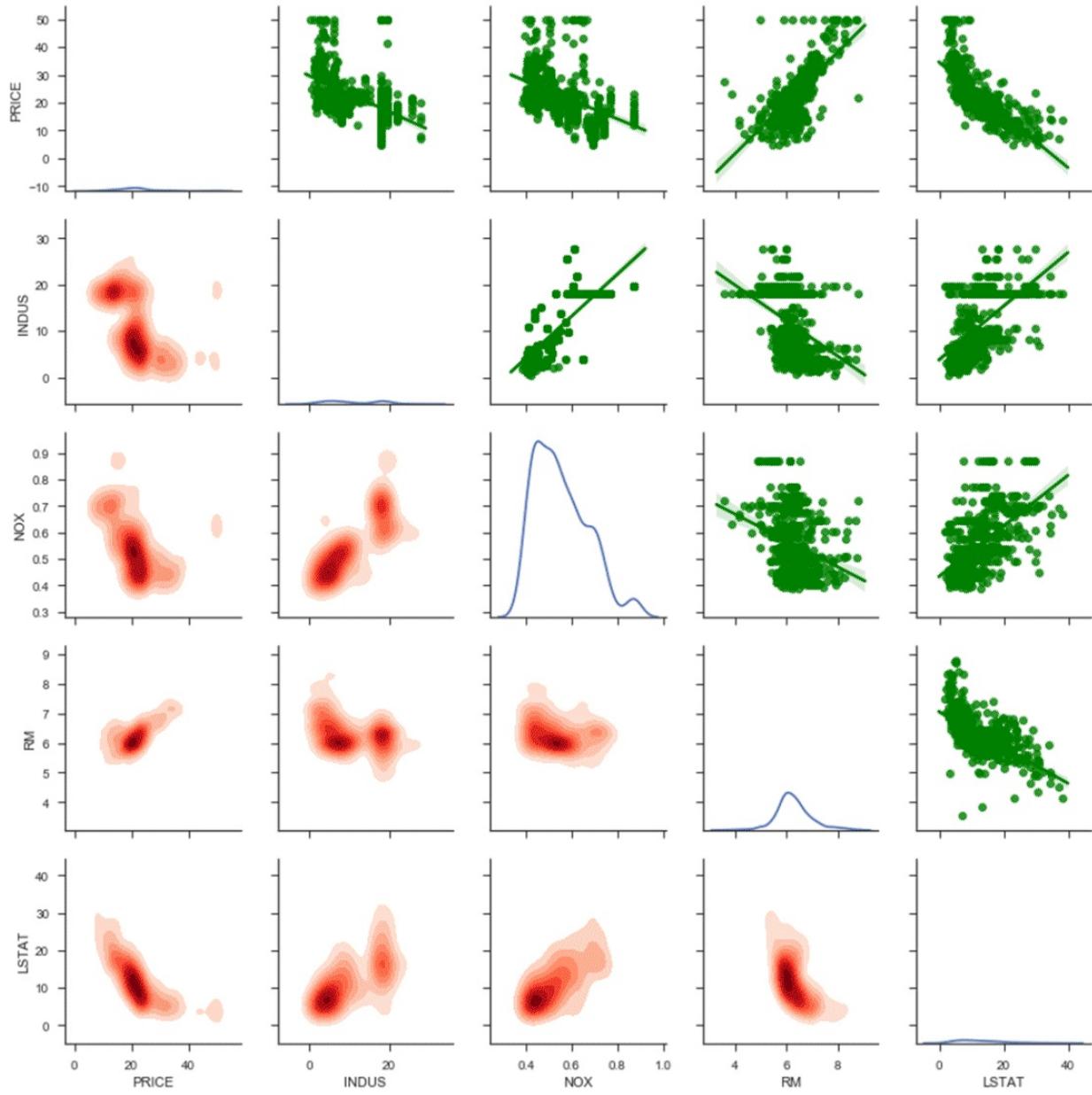
además, el PRECIO.

Nota que el error estándar se agrega de forma predeterminada al gráfico de dispersión.

```
attributes = ["PRICE", "INDUS", "NOX", "RM", "LSTAT"]

g = sns.PairGrid(df[attributes])
g = g.map_upper(sns.regplot, color="g")
g = g.map_lower(sns.kdeplot, cmap="Reds", shade=True,
shade_lowest=False)
g = g.map_diag(sns.kdeplot)
```

Salida



Comencemos con la parte superior:

- El precio está correlacionado negativamente con INDUS, NOX y LSTAT; correlacionado positivamente con RM.
- Hay una ligera no-linealidad con LSTAT y PRICE
- Hay como una línea recta cuando el precio es igual a 50. A partir de la descripción del conjunto de datos, PRICE se ha truncado en el valor de 50

Diagonal

- NOX parece tener dos racimos, uno alrededor de 0.5 y otro alrededor de 0.85.

Para verificar más al respecto, puede mirar la parte inferior. La densidad Multivariate Kernel es interesante en cierto sentido que colorea donde la mayoría de los puntos están. La diferencia con el gráfico de dispersión dibuja una densidad de probabilidad, aunque no hay ningún punto en el dataset para una coordenada dada. Cuando el color es más fuerte, indica una alta concentración de punto alrededor de esta área.

Si se comprueba la densidad multivariante para INDUS y NOX, se puede ver la correlación positiva y los dos clusters. Cuando la proporción de la industria es superior a 18, la concentración de óxidos nítricos es superior a 0.6.

Se puede pensar en agregar una interacción entre INDUS y NOX en la regresión lineal.

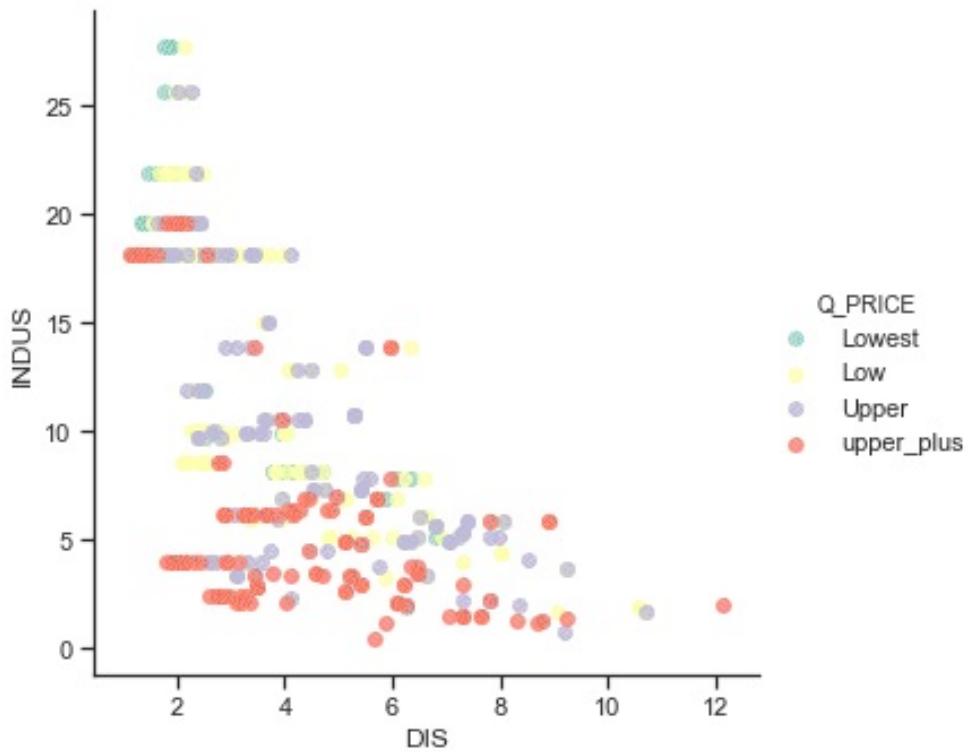
Finalmente, puede utilizar las segundas herramientas creadas por Google, Facets Deep Dive. La interfaz se divide en cuatro secciones principales. El área central del centro es una visualización ampliable de los datos. En la parte superior del panel, hay el menú desplegable donde puede cambiar la disposición de los datos para controlar facetado, posicionamiento y color. A la derecha, hay una vista detallada de una fila específica de datos. Esto significa que puede hacer clic en cualquier punto de datos en la visualización del centro para ver los detalles sobre ese punto de datos en particular.

Durante el paso de visualización de datos, usted está interesado en buscar la correlación entre la variable independiente sobre el precio de la casa. Sin embargo, implica al menos tres variables, y los trazados 3D son

complicados de trabajar.

Una forma de abordar este problema es crear una variable categórica. Es decir, podemos crear un trazado 2D de un color del punto. Puede dividir la variable PRECIO en cuatro categorías, con cada categoría es un cuartil (es decir, 0.25, 0.5, 0.75). Usted llama a esta nueva variable Q_PRICE.

```
## Check non linearity with important features
df['Q_PRICE'] = pd.qcut(df['PRICE'], 4, labels=["Lowest", "Low",
"Upper", "upper_plus"])
## Show non linearity between RM and LSTAT
ax = sns.lmplot(x="DIS", y="INDUS", hue="Q_PRICE", data=df, fit_reg
= False, palette="Set3")
```



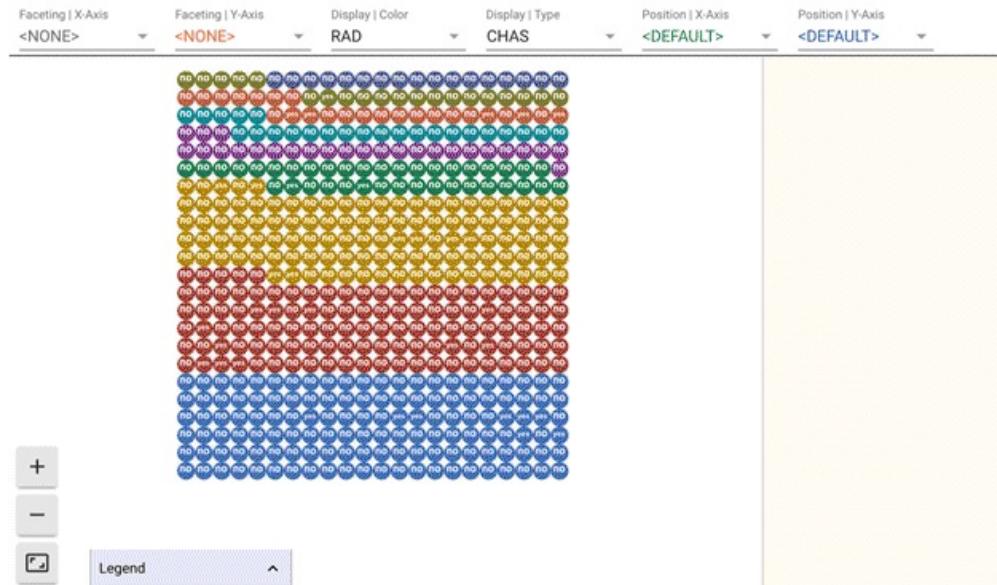
Inmersión profunda de facetas

Para abrir Deep Dive, debe transformar los datos en un formato json. Pandas como un objeto para eso. Puede usar `to_json` después del dataset Pandas.

La primera línea de código controla el tamaño del dataset.

```
df['Q_PRICE'] = pd.qcut(df['PRICE'], 4, labels=["Lowest", "Low",  
"Upper", "upper_plus"])  
sprite_size = 32 if len(df.index)>50000 else 64  
jsonstr = df.to_json(orient='records')
```

El siguiente código proviene de Google GitHub. Despues de ejecutar el código, debería poder ver esto:



```
# Display thde Dive visualization for this data  
from IPython.core.display import display, HTML  
  
# Create Facets template  
HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-  
dist/facets-jupyter.html">  
    <facets-dive sprite-image-width="{sprite_size}" sprite-
```

```

image-height="{sprite_size}" id="elem" height="600"></facets-dive>
<script>
    document.querySelector("#elem").data = {jsonstr};
</script>"""

# Load the json dataset and the sprite_size into the template
html = HTML_TEMPLATE.format(jsonstr=jsonstr,
sprite_size=sprite_size)

# Display the template
display(HTML(html))

```

Usted está interesado en ver si hay una conexión entre la tasa de la industria, la concentración de óxido, la distancia al centro de trabajo y el precio de la casa.

Para eso, primero divide los datos por rango de industria y color con el cuartilo de precio:

- Seleccione facetado X y elija INDUS.
- Seleccione Mostrar y elija DIS. Colorea los puntos con el cuartil del precio de la casa

aquí, los colores más oscuros significan que la distancia al primer centro de trabajo está lejos.

Hasta ahora, muestra de nuevo lo que sabes, menor tasa de industria, precio más alto. Ahora puede ver el desglose por INDUX, por NOX.

- Seleccione facetado Y y elija NOX.

Ahora se puede ver la casa lejos del primer centro de trabajo tiene la cuota más baja de la industria y, por lo tanto, la concentración de óxido más baja. Si elige mostrar el tipo con Q_PRICE y ampliar la esquina inferior izquierda, puede ver qué tipo de precio es.

Tiene otra pista de que la interacción entre IND, NOX y DIS puede ser

buenos candidatos para mejorar el modelo.

TensorFlow

En esta sección, calculará el clasificador lineal con la API de estimadores TensorFlow. Procederá de la siguiente manera:

- Preparar los datos
- Estimar un modelo de referencia: Sin interacción
- Estimar un modelo con interacción

Recuerde, el objetivo del aprendizaje automático es minimizar el error. En este caso, el modelo con el error cuadrado medio más bajo ganará. El estimador TensorFlow calcula automáticamente esta métrica.

Datos de preparación

En la mayoría de los casos, debe transformar sus datos. Es por eso que Facets Overview es fascinante. A partir de la estadística de resumen, vio que hay valores atípicos. Estos valores afectan a las estimaciones porque no se parecen a la población que está analizando. Los valores atípicos generalmente sesgados los resultados. Por ejemplo, un valor atípica positivo tiende a sobreestimar el coeficiente.

Una buena solución para abordar este problema es estandarizar la variable. Normalización significa una desviación estándar de uno y medios de cero. El proceso de estandarización implica dos pasos. En primer lugar, resta el valor medio de la variable. En segundo lugar, se divide por la varianza para que la distribución tenga una varianza de unidad

La biblioteca sklearn es útil para estandarizar variables. Puede utilizar el preprocesamiento del módulo con la escala de objetos para este

propósito.

Puede utilizar la siguiente función para escalar un dataset. Tenga en cuenta que no escala la columna de etiqueta ni las variables categóricas.

```
from sklearn import preprocessing
def standardize_data(df):
    X_scaled = preprocessing.scale(df[['CRIM', 'ZN', 'INDUS',
    'NOX', 'RM', 'AGE', 'DIS', 'RAD',
    'TAX', 'PTRATIO', 'B', 'LSTAT']])
    X_scaled_df = pd.DataFrame(X_scaled, columns = ['CRIM', 'ZN',
    'INDUS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
    'TAX', 'PTRATIO', 'B', 'LSTAT'])
    df_scale = pd.concat([X_scaled_df,
                          df['CHAS'],
                          df['PRICE']], axis=1, join='inner')
    return df_scale
```

Puede utilizar la función para construir el conjunto de tren/prueba escalado.

```
df_train_scale = standardize_data(df_train)
df_test_scale = standardize_data(df_test)
```

Regresión básica: Benchmark

En primer lugar, entrena y pruebas un modelo sin interacción. El propósito es ver la métrica de rendimiento del modelo.

La forma de entrenar el modelo es exactamente como el tutorial sobre **API de alto nivel**. Utilizará el estimador TensorFlow LinearRegorsor.

Como recordatorio, debe elegir:

- las características para poner en el modelo
- transformar las entidades
- construir el regresor lineal
- construir la función input_fn

- entrenar el modelo
- probar el modelo

Utilice todas las variables del dataset para entrenar el modelo. En total, hay variables continuas elevel y una variable categórica

```
## Add features to the bucket:
### Define continuous list
CONTI_FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
CATE_FEATURES = ['CHAS']
```

Convertir las entidades en una columna numérica o una columna categórica

```
continuous_features = [tf.feature_column.numeric_column(k) for k in
CONTI_FEATURES]
#categorical_features =
tf.feature_column.categorical_column_with_hash_bucket(CATE_FEATURES,
, hash_bucket_size=1000)
categorical_features =
[tf.feature_column.categorical_column_with_vocabulary_list('CHAS',
['yes', 'no'])]
```

Cree el modelo con LineArRegreso. Almacene el modelo en la carpeta train_Boston

```
model = tf.estimator.LinearRegressor(
    model_dir="train_Boston",
    feature_columns=categorical_features + continuous_features)
```

Salida

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train_Boston',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
```

```
0x1a19e76ac8>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

Cada columna de los datos de tren o prueba se convierte en un Tensor con la función get_input_fn

```
FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'CHAS']
LABEL= 'PRICE'
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

Estima el modelo en los datos del tren.

```
model.train(input_fn=get_input_fn(df_train_scale,
                                   num_epochs=None,
                                   n_batch = 128,
                                   shuffle=False),
            steps=1000)
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
train_Boston/model.ckpt.
INFO:tensorflow:loss = 56417.703, step = 1
INFO:tensorflow:global_step/sec: 144.457
INFO:tensorflow:loss = 76982.734, step = 101 (0.697 sec)
INFO:tensorflow:global_step/sec: 258.392
```

```
INFO:tensorflow:loss = 21246.334, step = 201 (0.383 sec)
INFO:tensorflow:global_step/sec: 227.998
INFO:tensorflow:loss = 30534.78, step = 301 (0.439 sec)
INFO:tensorflow:global_step/sec: 210.739
INFO:tensorflow:loss = 36794.5, step = 401 (0.477 sec)
INFO:tensorflow:global_step/sec: 234.237
INFO:tensorflow:loss = 8562.981, step = 501 (0.425 sec)
INFO:tensorflow:global_step/sec: 238.1
INFO:tensorflow:loss = 34465.08, step = 601 (0.420 sec)
INFO:tensorflow:global_step/sec: 237.934
INFO:tensorflow:loss = 12241.709, step = 701 (0.420 sec)
INFO:tensorflow:global_step/sec: 220.687
INFO:tensorflow:loss = 11019.228, step = 801 (0.453 sec)
INFO:tensorflow:global_step/sec: 232.702
INFO:tensorflow:loss = 24049.678, step = 901 (0.432 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
train_Boston/model.ckpt.
INFO:tensorflow:Loss for final step: 23228.568.
```

```
<tensorflow.python.estimator.canned.linear.LinearRegressor at
0x1a19e76320>
```

Por fin, se estima el rendimiento del modelo en el conjunto de pruebas

```
model.evaluate(input_fn=get_input_fn(df_test_scale,
                                      num_epochs=1,
                                      n_batch = 128,
                                      shuffle=False),
                      steps=1000)
```

Salida

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-29-02:40:43
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train_Boston/model.ckpt-
1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-29-02:40:43
INFO:tensorflow:Saving dict for global step 1000: average_loss =
86.89361, global_step = 1000, loss = 1650.9785
```

```
{'average_loss': 86.89361, 'global_step': 1000, 'loss': 1650.9785}
```

La pérdida del modelo es 1650. Esta es la métrica a batir en la siguiente sección

Mejorar el modelo: Término de interacción

Durante la primera parte del tutorial, vio una interesante relación entre las variables. Las diferentes técnicas de visualización revelaron que INDUS y NOS están vinculados entre sí y se convierte en magnificar el efecto sobre el precio. No sólo la interacción entre INDUS y NOS afecta el precio, sino que también este efecto es más fuerte cuando interactúa con DIS.

Es hora de generalizar esta idea y ver si se puede mejorar la predicción del modelo.

Debe agregar dos nuevas columnas a cada conjunto de conjuntos de datos: tren + prueba. Para ello, se crea una función para calcular el término de interacción y otra para calcular el término de interacción triple. Cada función produce una sola columna. Una vez creadas las nuevas variables, puede concatenarlas con el conjunto de datos de formación y el conjunto de datos de prueba.

En primer lugar, debe crear una nueva variable para la interacción entre INDUS y NOX.

La siguiente función devuelve dos dataframes, train y test, con la interacción entre var_1 y var_2, en su caso INDUS y NOX.

```
def interaction_term(var_1, var_2, name):
    t_train = df_train_scale[var_1]*df_train_scale[var_2]
    train = t_train.rename(name)
    t_test = df_test_scale[var_1]*df_test_scale[var_2]
    test = t_test.rename(name)
    return train, test
```

Almacena las dos nuevas columnas

```
interation_ind_ns_train, interation_ind_ns_test=
interaction_term('INDUS', 'NOX', 'INDUS_NOS')
interation_ind_ns_train.shape
(325, )
```

En segundo lugar, se crea una segunda función para calcular el término de interacción triple.

```
def triple_interaction_term(var_1, var_2, var_3, name):
    t_train =
df_train_scale[var_1]*df_train_scale[var_2]*df_train_scale[var_3]
    train = t_train.rename(name)
    t_test =
df_test_scale[var_1]*df_test_scale[var_2]*df_test_scale[var_3]
    test = t_test.rename(name)
    return train, test
interation_ind_ns_dis_train, interation_ind_ns_dis_test=
triple_interaction_term('INDUS', 'NOX', 'DIS', 'INDUS_NOS_DIS')
```

Ahora que tiene todas las columnas necesarias, puede agregarlas al conjunto de datos de entrenamiento y prueba. Nombra estos dos marcos de datos nuevos:

- df_train_new
- df_test_new

```
df_train_new = pd.concat([df_train_scale,
                           interation_ind_ns_train,
                           interation_ind_ns_dis_train],
                           axis=1, join='inner')
df_test_new = pd.concat([df_test_scale,
                        interation_ind_ns_test,
                        interation_ind_ns_dis_test],
                        axis=1, join='inner')
df_train_new.head(5)
```

Salida

	CRIM	ZN	INDUS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS	PRICE	INDUS_NOS
2	-0.275582	-0.47701	-0.464046	-0.162933	-0.188265	0.812916	0.105941	-0.661477	-0.616881	1.147718	0.444455	0.803221	no	34.7	0.075608
4	1.017983	-0.47701	0.992729	1.594192	-0.595967	0.987593	-0.907724	1.636106	1.502932	0.776192	-1.278797	1.504488	no	36.2	1.582601
5	-0.407050	-0.47701	-1.155868	-0.589166	1.036257	0.620414	-0.164461	-0.891235	-0.835358	-0.338387	0.444455	-1.025327	no	28.7	0.680999
8	-0.367794	-0.47701	-0.747795	-0.432591	-0.157121	0.798656	-0.346465	-0.201960	-0.616881	-0.524150	0.426162	1.182209	no	16.5	0.323489
9	1.832214	-0.47701	0.992729	1.246247	-2.699598	0.795092	-1.129861	1.636106	1.502932	0.776192	-0.779497	2.450575	no	18.9	1.237185

Eso es todo; puede estimar el nuevo modelo con los términos de interacción y ver cómo es la métrica de rendimiento.

```
CONTI_FEATURES_NEW = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT',
'INDUS_NOS', 'INDUS_NOS_DIS']
### Define categorical list
continuous_features_new = [tf.feature_column.numeric_column(k) for
k in CONTI_FEATURES_NEW]
model = tf.estimator.LinearRegressor(
    model_dir="train_Boston_1",
    feature_columns= categorical_features +
continuous_features_new)
```

Salida

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train_Boston_1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a1a5d5860>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

CÓDIGO

```
FEATURES = ['CRIM', 'ZN', 'INDUS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'INDUS_NOS',
'INDUS_NOS_DIS', 'CHAS']
LABEL= 'PRICE'
```

```

def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)

model.train(input_fn=get_input_fn(df_train_new,
                                   num_epochs=None,
                                   n_batch = 128,
                                   shuffle=False),
            steps=1000)

```

Salida

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
train_Boston_1/model.ckpt.
INFO:tensorflow:loss = 56417.703, step = 1
INFO:tensorflow:global_step/sec: 124.844
INFO:tensorflow:loss = 65522.3, step = 101 (0.803 sec)
INFO:tensorflow:global_step/sec: 182.704
INFO:tensorflow:loss = 15384.148, step = 201 (0.549 sec)
INFO:tensorflow:global_step/sec: 208.189
INFO:tensorflow:loss = 22020.305, step = 301 (0.482 sec)
INFO:tensorflow:global_step/sec: 213.855
INFO:tensorflow:loss = 28208.812, step = 401 (0.468 sec)
INFO:tensorflow:global_step/sec: 209.758
INFO:tensorflow:loss = 7606.877, step = 501 (0.473 sec)
INFO:tensorflow:global_step/sec: 196.618
INFO:tensorflow:loss = 26679.76, step = 601 (0.514 sec)
INFO:tensorflow:global_step/sec: 196.472
INFO:tensorflow:loss = 11377.163, step = 701 (0.504 sec)
INFO:tensorflow:global_step/sec: 172.82
INFO:tensorflow:loss = 8592.07, step = 801 (0.578 sec)
INFO:tensorflow:global_step/sec: 168.916
INFO:tensorflow:loss = 19878.56, step = 901 (0.592 sec)

```

```
INFO:tensorflow:Saving checkpoints for 1000 into  
train_Boston_1/model.ckpt.  
INFO:tensorflow:Loss for final step: 19598.387.
```

```
<tensorflow.python.estimator.canned.linear.LinearRegressor at  
0x1a1a5d5e10>
```

```
model.evaluate(input_fn=get_input_fn(df_test_new,  
                                      num_epochs=1,  
                                      n_batch = 128,  
                                      shuffle=False),  
               steps=1000)
```

Salida

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-29-02:41:14  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from  
train_Boston_1/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-29-02:41:14  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
79.78876, global_step = 1000, loss = 1515.9863  
  
{'average_loss': 79.78876, 'global_step': 1000, 'loss': 1515.9863}
```

La nueva pérdida es 1515. Con sólo añadir dos nuevas variables, usted fue capaz de disminuir la pérdida. Esto significa que puede hacer una mejor predicción que con el modelo de referencia.

Capítulo 15: Clasificador lineal en TensorFlow

¿Qué es el clasificador lineal?

Las dos tareas de aprendizaje supervisado más comunes son la regresión lineal y el clasificador lineal. La regresión lineal predice un valor mientras que el clasificador lineal predice una clase. Este tutorial se centra en Linear Classifier.

Los problemas de clasificación representan aproximadamente el 80 por ciento de la tarea de aprendizaje automático. La clasificación tiene como objetivo predecir la probabilidad de cada clase dada un conjunto de entradas. La etiqueta (es decir, la variable dependiente) es un valor discreto, llamado clase.

1. Si la etiqueta tiene sólo dos clases, el algoritmo de aprendizaje es un clasificador binario.
2. El clasificador multiclass aborda etiquetas con más de dos clases.

Por ejemplo, un problema típico de clasificación binaria es predecir la probabilidad de que un cliente realice una segunda compra. Predecir el tipo de animal que se muestra en una imagen es problema de clasificación multiclase ya que hay más de dos variedades de animales existentes.

La parte teórica de este tutorial pone el foco principal en la clase binaria. Aprenderá más sobre la función de salida multiclase en un futuro tutorial.

¿Cómo funciona el clasificador binario?

Aprendió en el tutorial anterior que una función se compone de dos tipos de variables, una variable dependiente y un conjunto de entidades (variables independientes). En la regresión lineal, una variable dependiente es un número real sin rango. El objetivo principal es predecir su valor minimizando el error cuadrado medio.

Para una tarea binaria, la etiqueta puede haber tenido dos posibles valores enteros. En la mayoría de los casos, es [0,1] o [1,2]. Por ejemplo, el objetivo es predecir si un cliente va a comprar un producto o no. La etiqueta se define de la siguiente manera:

- $Y = 1$ (el cliente compró el producto)
- $Y = 0$ (el cliente no compra el producto)

El modelo utiliza las características X para clasificar a cada cliente en la clase más probable a la que pertenece, es decir, comprador potencial o no.

La probabilidad de éxito se calcula con **regresión logística**. El algoritmo calculará una probabilidad basada en la característica X y predice un éxito cuando esta probabilidad está por encima del 50 por ciento. Más formalmente, la probabilidad se calcula de la siguiente manera:

$$P(Y = 1|x) = \frac{1}{1 + \exp(-(\theta^T x + b))}$$

donde θ es el conjunto de pesos, las características y b el sesgo.

La función se puede descomponer en dos partes:

- El modelo lineal
- La función logística

Modelo lineal

Ya está familiarizado con la forma en que se calculan los pesos. Los pesos se calculan utilizando un producto de punto: $\theta^T x + b$ that is $\sum_{i=0}^n x_i w_i + b$. Y es una función lineal de todas las características x_i . Si el modelo no tiene entidades, la predicción es igual al sesgo, b .

Los pesos indican la dirección de la correlación entre las entidades x_i y la etiqueta y . Una correlación positiva aumenta la probabilidad de la clase positiva mientras que una correlación negativa lleva la probabilidad más cercana a 0, (es decir, clase negativa).

El modelo lineal devuelve sólo el número real, lo que es inconsistente con la medida de probabilidad del rango [0,1]. La función logística es necesaria para convertir la salida del modelo lineal en una probabilidad,

Función logística

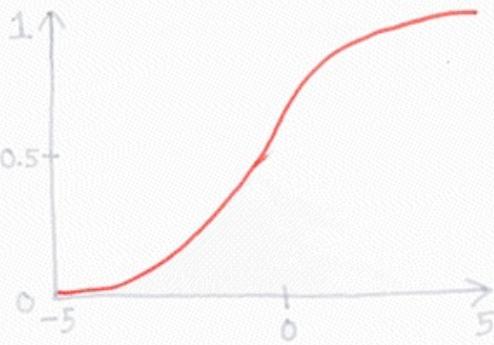
La función logística, o función sigmoide, tiene una forma de S y la salida de esta función siempre está entre 0 y 1.

$$\frac{1}{1 + \exp(-t)}$$

LOGISTIC SIGMOID FUNCTION

$$\sigma(x) = \frac{1}{1 + e^{(-x)}}$$

Input



Chris Albon

Es fácil sustituir la salida de la regresión lineal en la función sigmoide. Se traduce en un nuevo número con una probabilidad entre 0 y 1.

El clasificador puede transformar la probabilidad en una clase

- Los valores entre 0 y 0.49 se convierten en clase 0
- Los valores entre 0.5 y 1 se convierten en clase 1

¿Cómo medir el rendimiento de Linear Classifier?

Exactitud

El rendimiento general de un clasificador se mide con la métrica de precisión. Precisión recopila todos los valores correctos divididos por el número total de observaciones. Por ejemplo, un valor de precisión del 80 por ciento significa que el modelo es correcto en el 80 por ciento de los casos.

ACCURACY

$$A_{cc} = \frac{1}{n} \sum \mathbf{1}(\hat{y}_i = y_i)$$

Diagrama explicativo de la fórmula de Accuracy:

- Predicción y**: Etiqueta para la variable \hat{y}_i .
- Verdadero y**: Etiqueta para la variable y_i .
- Función de indicador**: Etiqueta para la función $\mathbf{1}$.
- número de observaciones**: Etiqueta para el denominador n .

Puede notar una deficiencia con esta métrica, especialmente para la clase de desequilibrio. Un dataset de desequilibrio se produce cuando el número de observaciones por grupo no es igual. Digamos, tratás de clasificar un evento raro con una función logística. Imagine que el clasificador intenta estimar la muerte de un paciente después de una enfermedad. En los datos, el 5 por ciento de los pacientes fallan. Puede entrenar a un clasificador para predecir el número de muertes y utilizar la

métrica de precisión para evaluar las actuaciones. Si el clasificador predice o muerte para todo el conjunto de datos, será correcto en el 95 por ciento del caso.

Matriz de confusión

Una mejor manera de evaluar el rendimiento de un clasificador es mirar la matriz de confusión.

		Predicted	
		TRUE	FALSE
Actual	TRUE	TP	FN
	FALSE	FP	TN

La matriz de confusión visualiza la precisión de un clasificador comparando las clases reales y predecidas. La matriz binaria de confusión se compone de cuadrados:

- TP: Verdadero positivo: valores predecidos correctamente como positivos reales
- FP: Los valores predichos predijeron incorrectamente un positivo real. Es decir, los valores negativos predichos como positivos
- FN: Falso negativo: valores positivos pronosticados como negativos
- TN: Negativo Verdadero: Valores pronosticados correctamente como negativo real

A partir de la matriz de confusión, es fácil comparar la clase real y la clase predecible.

Precisión y sensibilidad

La matriz de confusión proporciona una buena visión de lo verdadero positivo y falso positivo. En algún caso, es preferible tener una métrica más concisa.

Precisión

La métrica de precisión muestra la precisión de la clase positiva. Mide la probabilidad de que la predicción de la clase positiva es correcta.

$$Precision = \frac{TP}{TP + FP}$$

La puntuación máxima es 1 cuando el clasificador clasifica perfectamente todos los valores positivos. La precisión por sí sola no es muy útil porque ignora la clase negativa. La métrica suele emparejarse con la métrica de recuperación. Recordemos también se llama sensibilidad o tasa positiva verdadera.

Sensibilidad

La sensibilidad calcula la relación de clases positivas detectadas correctamente. Esta métrica indica lo bueno que es el modelo para reconocer una clase positiva.

$$Recall = \frac{TP}{TP + FN}$$

Clasificador lineal con TensorFlow

Para este tutorial, vamos a utilizar el conjunto de datos censales. El propósito es utilizar las variables del conjunto de datos censales para predecir el nivel de ingresos. Tenga en cuenta que el ingreso es una variable binaria

- con un valor de 1 si los ingresos > 50k
- o si los ingresos son < 50k.

Esta variable es su etiqueta

Este conjunto de datos incluye ocho variables categóricas:

- lugar de trabajo
- educación
- conyugal
- ocupación
- relación
- raza
- relaciones sexuales
- país_nativo

además, seis variables continuas:

- edad
- Fnlwgt
- num_de_educación
- ganancia_capital
- capital_pérdida
- horas_semana

A través de este ejemplo, comprenderá cómo entrenar un clasificador lineal con el estimador TensorFlow y cómo mejorar la métrica de precisión.

Procederemos de la siguiente manera:

- Paso 1) Importar los datos
- Paso 2) Conversión de datos
- Paso 3) Entrenar al clasificador
- Paso 4) Mejorar el modelo
- Paso 5) Hiperparámetro: Lazo y cresta

Paso 1) Importar los datos

Primero importará las bibliotecas utilizadas durante el aprendizaje.

```
import tensorflow as tf  
import pandas as pd
```

A continuación, importará los datos del archivo de UCI y definirá los nombres de las columnas. Utilizará las COLUMNAS para nombrar las columnas en un marco de datos pandas.

Tenga en cuenta que entrenará el clasificador usando un marco de datos Pandas.

```
## Define path data  
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',  
'education_num', 'marital',  
          'occupation', 'relationship', 'race', 'sex',  
'capital_gain', 'capital_loss',  
          'hours_week', 'native_country', 'label']  
PATH = "https://archive.ics.uci.edu/ml/machine-learning-  
databases/adult/adult.data"  
PATH_test = "https://archive.ics.uci.edu/ml/machine-learning-  
databases/adult/adult.test"
```

Los datos almacenados en línea ya están divididos entre un conjunto de trenes y un conjunto de pruebas.

```
df_train = pd.read_csv(PATH, skipinitialspace=True, names =  
COLUMNS, index_col=False)  
df_test = pd.read_csv(PATH_test, skiprows = 1,  
skipinitialspace=True, names = COLUMNS, index_col=False)
```

El conjunto de trenes contiene 32.561 observaciones y el conjunto de pruebas 16.281

```
print(df_train.shape, df_test.shape)  
print(df_train.dtypes)  
(32561, 15) (16281, 15)  
age           int64  
workclass     object  
fnlwgt        int64  
education     object  
education_num int64  
marital       object  
occupation    object  
relationship   object  
race          object  
sex           object  
capital_gain  int64  
capital_loss  int64  
hours_week    int64  
native_country object  
label         object  
dtype: object
```

Tensorflow requiere un valor booleana para entrenar el clasificador. Debe convertir los valores de cadena a entero. La etiqueta se almacena como un objeto, sin embargo, debe convertirla en un valor numérico. El siguiente código crea un diccionario con los valores para convertir y bucle sobre el elemento de columna. Tenga en cuenta que realiza esta operación dos veces, una para la prueba del tren, otra para el conjunto de pruebas

```
label = {'<=50K': 0, '>50K': 1}  
df_train.label = [label[item] for item in df_train.label]
```

```
label_t = {'<=50K.': 0, '>50K.': 1}
df_test.label = [label_t[item] for item in df_test.label]
```

En los datos del tren, hay 24,720 ingresos inferiores a 50 mil y 7841 más arriba. La relación es casi la misma para el conjunto de pruebas. Consulte este tutorial sobre Facetas para obtener más información.

```
print(df_train["label"].value_counts())
### The model will be correct in atleast 70% of the case
print(df_test["label"].value_counts())
## Unbalanced label
print(df_train.dtypes)
0    24720
1    7841
Name: label, dtype: int64
0    12435
1    3846
Name: label, dtype: int64
age            int64
workclass      object
fnlwgt          int64
education      object
education_num   int64
marital         object
occupation     object
relationship   object
race            object
sex             object
capital_gain   int64
capital_loss   int64
hours_week      int64
native_country  object
label           int64
dtype: object
```

Paso 2) Conversión de datos

Se requieren algunos pasos antes de entrenar un clasificador lineal con Tensorflow. Debe preparar las funciones que se incluirán en el modelo. En la regresión de referencia, utilizará los datos originales sin aplicar

ninguna transformación.

El estimador necesita tener una lista de funciones para entrenar el modelo. Por lo tanto, los datos de la columna requieren ser convertidos en un tensor.

Una buena práctica consiste en definir dos listas de entidades según su tipo y luego pasarlas en las columnas_feature del estimador.

Comenzará convirtiendo entidades continuas y, a continuación, definirá un depósito con los datos categóricos.

Las características del dataset tienen dos formatos:

- Entero
- Objeto

Cada entidad se enumera en las dos variables siguientes según sus tipos.

```
## Add features to the bucket:  
### Define continuous list  
CONTI_FEATURES = ['age', 'fnlwgt', 'capital_gain', 'education_num',  
'capital_loss', 'hours_week']  
### Define the categorical list  
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',  
'relationship', 'race', 'sex', 'native_country']
```

El feature_column está equipado con un objeto numeric_column para ayudar en la transformación de las variables continuas en tensor. En el siguiente código, se convierten todas las variables de CONTI_Features en un tensor con un valor numérico. Esto es obligatorio para construir el modelo. Todas las variables independientes deben convertirse en el tipo adecuado de tensor.

A continuación escribimos un código que le permitirá ver lo que está sucediendo detrás de feature_column.numeric_column. Imprimiremos

el valor convertido para la edad Es con fines explicativos, por lo tanto, no hay necesidad de entender el código python. Puede consultar la documentación oficial para entender los códigos.

```
def print_transformation(feature = "age", continuous = True, size = 2):
    #X = fc.numeric_column(feature)
    ## Create feature name
    feature_names = [
        feature]

    ## Create dict with the data
    d = dict(zip(feature_names, [df_train[feature]]))

    ## Convert age
    if continuous == True:
        c = tf.feature_column.numeric_column(feature)
        feature_columns = [c]
    else:
        c =
    tf.feature_column.categorical_column_with_hash_bucket(feature,
    hash_bucket_size=size)
        c_indicator = tf.feature_column.indicator_column(c)
        feature_columns = [c_indicator]

    ## Use input_layer to print the value
    input_layer = tf.feature_column.input_layer(
        features=d,
        feature_columns=feature_columns
    )
    ## Create lookup table
    zero = tf.constant(0, dtype=tf.float32)
    where = tf.not_equal(input_layer, zero)
    ## Return lookup tble
    indices = tf.where(where)
    values = tf.gather_nd(input_layer, indices)
    ## Initiate graph
    sess = tf.Session()
    ## Print value
    print(sess.run(input_layer))
print_transformation(feature = "age", continuous = True)
[[39.]
 [50.]]
```

```
[38.]  
...  
[58.]  
[22.]  
[52.]]
```

Los valores son exactamente los mismos que en df_train

```
continuous_features = [tf.feature_column.numeric_column(k) for k in  
CONTI_FEATURES]
```

Según la documentación de TensorFlow, hay diferentes formas de convertir datos categóricos. Si la lista de vocabulario de una función es conocida y no tiene muchos valores, es posible crear la columna categórica con categorical_column_with_vocabulary_list. Se asignará a toda la lista de vocabulario único un ID.

Por ejemplo, si un estado de variable tiene tres valores distintos:

- Marido
- Esposa
- Soltero

Entonces se le atribuirán tres ID. Por ejemplo, el marido tendrá el ID 1, la esposa el ID 2 y así sucesivamente.

Para fines ilustrativos, puede utilizar este código para convertir una variable de objeto en una columna categórica en TensorFlow.

La característica sexual sólo puede tener dos valores: masculino o femenino. Cuando vamos a convertir la característica sexual, Tensorflow creará 2 nuevas columnas, una para hombre y otra para mujer. Si el género es igual a masculino, entonces la nueva columna masculina será igual a 1 y hembra a 0. Este ejemplo se muestra en la tabla siguiente:

filas	relaciones sexuales	después de la transformación	masculino	femenino
-------	---------------------	------------------------------	-----------	----------

1	masculino	=>	1	0
2	masculino	=>	1	0
3	femenino	=>	0	1

En tensorflow:

```
print_transformation(feature = "sex", continuous = False, size = 2)
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [1. 0.]
 [0. 1.]]]

relationship =
tf.feature_column.categorical_column_with_vocabulary_list(
    'relationship', [
        'Husband', 'Not-in-family', 'Wife', 'Own-child',
    'Unmarried',
    'Other-relative'])
```

A continuación, agregamos código Python para imprimir la codificación. Una vez más, no necesita entender el código, el propósito es ver la transformación

Sin embargo, una forma más rápida de transformar los datos es utilizar el método `categorical_column_with_hash_bucket`. Alterar variables de cadena en una matriz dispersa será útil. Una matriz dispersa es una matriz con mayormente cero. El método se encarga de todo. Solo necesita especificar el número de cubos y la columna clave. El número de cubos es la cantidad máxima de grupos que Tensorflow puede crear. La columna clave es simplemente el nombre de la columna que se va a convertir.

En el siguiente código, se crea un bucle sobre todas las entidades categóricas.

```
categorical_features =  
[tf.feature_column.categorical_column_with_hash_bucket(k,  
hash_bucket_size=1000) for k in CATE_FEATURES]
```

Paso 3) Entrena al Clasificador

TensorFlow proporciona actualmente un estimador para la regresión lineal y la clasificación lineal.

- Regresión lineal: LinealRegresor
- Clasificación lineal: LinearClassifier

La sintaxis del clasificador lineal es la misma que en el tutorial sobre regresión lineal excepto por un argumento, n_class. Debe definir la columna de elemento, el directorio del modelo y, comparar con el regresor lineal; tiene el definir el número de clase. Para una regresión logit, el número de clase es igual a 2.

El modelo calculará los pesos de las columnas contenidas en continuous_features y categorical_features.

```
model = tf.estimator.LinearClassifier(  
    n_classes = 2,  
    model_dir="ongoing/train",  
    feature_columns=categorical_features+ continuous_features)
```

SALIDA:

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,  
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':  
100, '_train_distribute': None, '_service': None, '_cluster_spec':
```

```
<tensorflow.python.training.server_lib.ClusterSpec object at  
0x181f24c898>, '_task_type': 'worker', '_task_id': 0,  
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':  
'', '_is_chief': True, '_num_ps_replicas': 0,  
'_num_worker_replicas': 1}
```

Ahora que el clasificador está definido, puede crear la función de entrada. El método es el mismo que en el tutorial regresor lineal. Aquí, se utiliza un tamaño de lote de 128 y se barajan los datos.

```
FEATURES = ['age', 'workclass', 'fnlwgt', 'education',  
'education_num', 'marital', 'occupation', 'relationship', 'race',  
'sex', 'capital_gain', 'capital_loss', 'hours_week',  
'native_country']  
LABEL= 'label'  
def get_input_fn(data_set, num_epochs=None, n_batch = 128,  
shuffle=True):  
    return tf.estimator.inputs.pandas_input_fn(  
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),  
        y = pd.Series(data_set[LABEL].values),  
        batch_size=n_batch,  
        num_epochs=num_epochs,  
        shuffle=shuffle)
```

Se crea una función con los argumentos requeridos por el estimador lineal, es decir, número de épocas, número de lotes y barajar el dataset o la nota. Dado que utiliza el método Pandas para pasar los datos al modelo, debe definir las variables X como un marco de datos pandas. Tenga en cuenta que recorre todos los datos almacenados en CARACTERÍSTICAS.

Vamos a entrenar el modelo con el `model.train` objeto. Utilice la función previamente definida para alimentar el modelo con los valores adecuados. Tenga en cuenta que establece el tamaño del lote en 128 y el número de épocas en Ninguno. El modelo será entrenado más de mil pasos.

```
model.train(input_fn=get_input_fn(df_train,  
                                    num_epochs=None,
```

```
n_batch = 128,  
shuffle=False),  
steps=1000)
```

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow>Create CheckpointSaverHook.  
INFO:tensorflow: Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into  
ongoing/train/model.ckpt.  
INFO:tensorflow:loss = 88.722855, step = 1  
INFO:tensorflow:global_step/sec: 65.8282  
INFO:tensorflow:loss = 52583.64, step = 101 (1.528 sec)  
INFO:tensorflow:global_step/sec: 118.386  
INFO:tensorflow:loss = 25203.816, step = 201 (0.837 sec)  
INFO:tensorflow:global_step/sec: 110.542  
INFO:tensorflow:loss = 54924.312, step = 301 (0.905 sec)  
INFO:tensorflow:global_step/sec: 199.03  
INFO:tensorflow:loss = 68509.31, step = 401 (0.502 sec)  
INFO:tensorflow:global_step/sec: 167.488  
INFO:tensorflow:loss = 9151.754, step = 501 (0.599 sec)  
INFO:tensorflow:global_step/sec: 220.155  
INFO:tensorflow:loss = 34576.06, step = 601 (0.453 sec)  
INFO:tensorflow:global_step/sec: 199.016  
INFO:tensorflow:loss = 36047.117, step = 701 (0.503 sec)  
INFO:tensorflow:global_step/sec: 197.531  
INFO:tensorflow:loss = 22608.148, step = 801 (0.505 sec)  
INFO:tensorflow:global_step/sec: 208.479  
INFO:tensorflow:loss = 22201.918, step = 901 (0.479 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into  
ongoing/train/model.ckpt.  
INFO:tensorflow:Loss for final step: 5444.363.  
  
<tensorflow.python.estimator.canned.linear.LinearClassifier at  
0x181f223630>
```

Tenga en cuenta que la pérdida disminuyó posteriormente durante los últimos 100 pasos, es decir, de 901 a 1000.

La pérdida final después de mil iteraciones es 5444. Puede estimar su

modelo en el conjunto de pruebas y ver el rendimiento. Para evaluar el rendimiento del modelo, debe utilizar el objeto evaluar. Alimenta el modelo con el conjunto de pruebas y establece el número de épocas en 1, es decir, los datos irán al modelo sólo una vez.

```
model.evaluate(input_fn=get_input_fn(df_test,
                                      num_epochs=1,
                                      n_batch = 128,
                                      shuffle=False),
                           steps=1000)
```

```
INFO:tensorflow:Calling model_fn.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:22  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from ongoing/train/model.ckpt-  
1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [100/1000]  
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:23  
INFO:tensorflow:Saving dict for global step 1000: accuracy =  
0.7615626, accuracy_baseline = 0.76377374, auc = 0.63300294,  
auc_precision_recall = 0.50891197, average_loss = 47.12155,  
global_step = 1000, label/mean = 0.23622628, loss = 5993.6406,  
precision = 0.49401596, prediction/mean = 0.18454961, recall =  
0.38637546  
  
{'accuracy': 0.7615626,  
 'accuracy_baseline': 0.76377374,  
 'auc': 0.63300294,  
 'auc_precision_recall': 0.50891197,  
 'average_loss': 47.12155,  
 'global_step': 1000,  
 'label/mean': 0.23622628,  
 'loss': 5993.6406,  
 'precision': 0.49401596,  
 'prediction/mean': 0.18454961},
```

```
'recall': 0.38637546}
```

TensorFlow devuelve todas las métricas que aprendiste en la parte teórica. Sin sorpresa, la precisión es grande debido a la etiqueta desequilibrada. En realidad, el modelo funciona ligeramente mejor que una suposición aleatoria. Imagine que el modelo predice todos los hogares con ingresos inferiores a 50K, entonces el modelo tiene una precisión del 70 por ciento. En un análisis más detallado, puede ver que la predicción y la recuperación son bastante bajos.

Paso 4) Mejorar el modelo

Ahora que tiene un modelo de referencia, puede intentar mejorarlo, es decir, aumentar la precisión. En el tutorial anterior, aprendió a mejorar la potencia de predicción con un término de interacción. En este tutorial, volverá a visitar esta idea agregando un término polinómico a la regresión.

La regresión polinomial es instrumental cuando hay no linealidad en los datos. Hay dos formas de capturar la no linealidad en los datos.

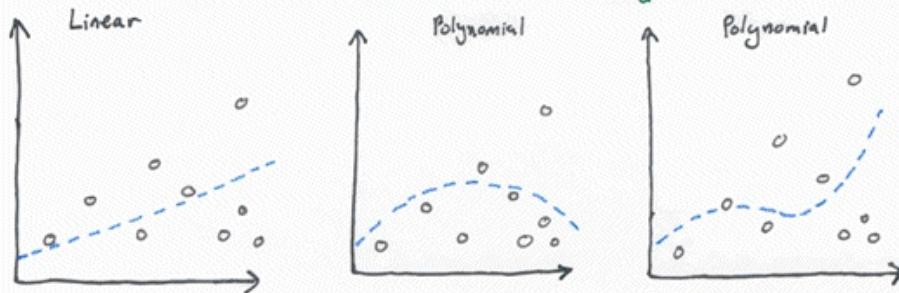
- Agregar término polinómico
- Bucketize la variable continua en una variable categórica

Término polinómico

En la imagen de abajo, puede ver qué es una regresión polinómica. Es una ecuación con variables X con diferente poder. Una regresión polinomial de segundo grado tiene dos variables, X y X al cuadrado. Tercer grado tiene tres variables, X, X^2 , y X^3

Polyhomial REGRESSION

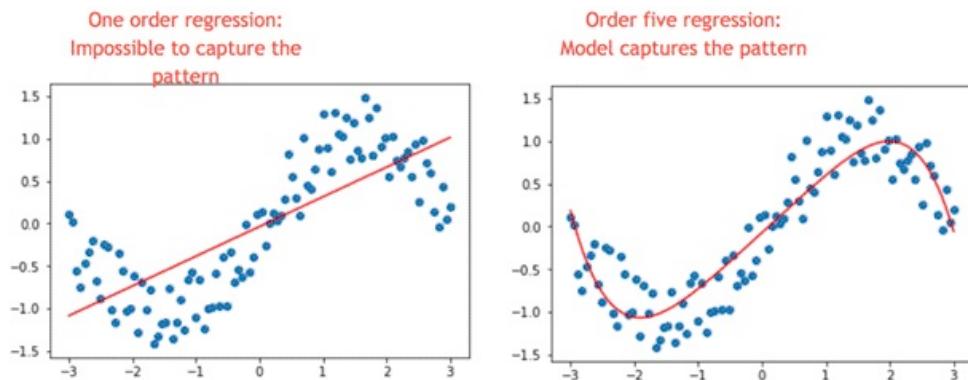
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \dots + \beta_d x_i^d + e_i$$



BY CHRIS ALBON

A continuación, construimos un gráfico con dos variables, X e Y. Es obvio que la relación no es lineal. Si añadimos una regresión lineal, podemos ver que el modelo no puede capturar el patrón (imagen izquierda).

Ahora, mira la imagen de la izquierda de la imagen de abajo, agregamos cinco términos a la regresión (que es $y=x+x^2+x^3+x^4+x^5$). El modelo ahora captura mucho mejor el patrón. Este es el poder de la regresión polinómica.



Volvamos a nuestro ejemplo. La edad no está en una relación lineal con los ingresos. La edad temprana puede tener un ingreso fijo cercano a cero

porque los niños o los jóvenes no trabajan. Luego aumenta en edad de trabajar y disminuye durante la jubilación. Típicamente es una forma de U invertida. Una forma de capturar este patrón es agregando un poder dos a la regresión.

Veamos si aumenta la precisión.

Debe agregar esta nueva entidad al dataset y en la lista de entidades continuas.

Agregue la nueva variable en el conjunto de datos de tren y prueba, por lo que es más conveniente escribir una función.

```
def square_var(df_t, df_te, var_name = 'age'):
    df_t['new'] = df_t[var_name].pow(2)
    df_te['new'] = df_te[var_name].pow(2)
    return df_t, df_te
```

La función tiene 3 argumentos:

- df_t: definir el conjunto de entrenamiento
- df_te: definir el conjunto de pruebas
- var_name = 'age': Define la variable a transformar

Puede usar el objeto pow (2) para cuadrar la edad variable. Tenga en cuenta que la nueva variable se llama 'new'

Ahora que la función square_var está escrita, puede crear los nuevos datasets.

```
df_train_new, df_test_new = square_var(df_train, df_test, var_name
= 'age')
```

Como puede ver, el nuevo dataset tiene una entidad más.

```
print(df_train_new.shape, df_test_new.shape)
(32561, 16) (16281, 16)
```

La variable cuadrada se llama nueva en el dataset. Debe agregarlo a la lista de entidades continuas.

```
CONTI_FEATURES_NEW = ['age', 'fnlwgt', 'capital_gain',
'education_num', 'capital_loss', 'hours_week', 'new']
continuous_features_new = [tf.feature_column.numeric_column(k) for
k in CONTI_FEATURES_NEW]
```

Nota que ha cambiado el directorio del gráfico. No puedes entrenar diferentes modelos en el mismo directorio. Significa que debe cambiar la ruta del argumento model_dir. Si no lo hace, TensorFlow arrojará un error.

```
model_1 = tf.estimator.LinearClassifier(
    model_dir="ongoing/train1",
    feature_columns=categorical_features+ continuous_features_new)
```

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec object at 0x1820f04b70>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master': '',
'_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
FEATURES_NEW = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_week',
'native_country', 'new']
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in
FEATURES_NEW}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

Ahora que el clasificador está diseñado con el nuevo conjunto de datos, puede entrenar y evaluar el modelo.

```
model_1.train(input_fn=get_input_fn(df_train,
                                     num_epochs=None,
                                     n_batch = 128,
                                     shuffle=False),
               steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
ongoing/train1/model.ckpt.
INFO:tensorflow:loss = 88.722855, step = 1
INFO:tensorflow:global_step/sec: 81.487
INFO:tensorflow:loss = 70077.66, step = 101 (1.228 sec)
INFO:tensorflow:global_step/sec: 111.169
INFO:tensorflow:loss = 49522.082, step = 201 (0.899 sec)
INFO:tensorflow:global_step/sec: 128.91
INFO:tensorflow:loss = 107120.57, step = 301 (0.776 sec)
INFO:tensorflow:global_step/sec: 132.546
INFO:tensorflow:loss = 12814.152, step = 401 (0.755 sec)
INFO:tensorflow:global_step/sec: 162.194
INFO:tensorflow:loss = 19573.898, step = 501 (0.617 sec)
INFO:tensorflow:global_step/sec: 204.852
INFO:tensorflow:loss = 26381.986, step = 601 (0.488 sec)
INFO:tensorflow:global_step/sec: 188.923
INFO:tensorflow:loss = 23417.719, step = 701 (0.529 sec)
INFO:tensorflow:global_step/sec: 192.041
INFO:tensorflow:loss = 23946.049, step = 801 (0.521 sec)
INFO:tensorflow:global_step/sec: 197.025
INFO:tensorflow:loss = 3309.5786, step = 901 (0.507 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train1/model.ckpt.
INFO:tensorflow:Loss for final step: 28861.898.

<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x1820f04c88>
```

```
model_1.evaluate(input_fn=get_input_fn(df_test_new,
```

```
    num_epochs=1,  
    n_batch = 128,  
    shuffle=False),  
    steps=1000)
```

```
INFO:tensorflow:Calling model_fn.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:37  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from  
ongoing/train1/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [100/1000]  
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:39  
INFO:tensorflow:Saving dict for global step 1000: accuracy =  
0.7944229, accuracy_baseline = 0.76377374, auc = 0.6093755,  
auc_precision_recall = 0.54885805, average_loss = 111.0046,  
global_step = 1000, label/mean = 0.23622628, loss = 14119.265,  
precision = 0.6682401, prediction/mean = 0.09116262, recall =  
0.2576703  
  
{'accuracy': 0.7944229,  
 'accuracy_baseline': 0.76377374,  
 'auc': 0.6093755,  
 'auc_precision_recall': 0.54885805,  
 'average_loss': 111.0046,  
 'global_step': 1000,  
 'label/mean': 0.23622628,  
 'loss': 14119.265,  
 'precision': 0.6682401,  
 'prediction/mean': 0.09116262,  
 'recall': 0.2576703}
```

La variable cuadrada mejoró la precisión de 0.76 a 0.79. Vamos a ver si se puede hacer mejor combinando la bucketización y el término de interacción juntos.

Bucketización e interacción

Como ha visto antes, un clasificador lineal no puede capturar correctamente el patrón de ingresos por edad. Esto se debe a que aprende un solo peso para cada característica. Para que sea más fácil para el clasificador, una cosa que puede hacer es cambiar la función. El bucketing transforma una entidad numérica en varias determinadas en función del rango en el que cae, y cada una de estas nuevas entidades indica si la edad de una persona está dentro de ese rango.

Con estas nuevas características, el modelo lineal puede capturar la relación aprendiendo diferentes pesos para cada cubo.

En TensorFlow, se realiza con `bucketized_column`. Debe agregar el rango de valores en los límites.

```
age = tf.feature_column.numeric_column('age')
age_buckets = tf.feature_column.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

Ya sabes que la edad no es lineal con los ingresos. Otra forma de mejorar el modelo es a través de la interacción. En la palabra de TensorFlow, es el cruce de entidades. El cruce de entidades es una forma de crear nuevas entidades que son combinaciones de las existentes, lo que puede ser útil para un clasificador lineal que no puede modelar interacciones entre entidades.

Puedes desglosar la edad con otra característica como la educación. Es decir, es probable que algunos grupos tengan un ingreso alto y otros bajos (Piense en el estudiante de doctorado).

```
education_x_occupation = [tf.feature_column.crossed_column(
    ['education', 'occupation'], hash_bucket_size=1000)]
age_buckets_x_education_x_occupation =
[tf.feature_column.crossed_column(
    [age_buckets, 'education', 'occupation'],
    hash_bucket_size=1000)]
```

Para crear una columna de entidad cruzada, utilice crossed_column con las variables que se cruzan en un paréntesis. El hash_bucket_size indica las posibilidades de cruce máximas. Para crear interacción entre variables (al menos una variable debe ser categórica), puede utilizar tf.feature_column.crossed_column. Para utilizar este objeto, debe agregar entre corchetes la variable para interactuar y un segundo argumento, el tamaño del cubo. El tamaño del cubo es el número máximo de grupo posible dentro de una variable. Aquí lo establece en 1000 ya que no sabe el número exacto de grupos

age_buckets necesita ser cuadrado antes de agregarlo a las columnas de entidades. También puede agregar las nuevas entidades a las columnas de entidades y preparar el estimador

```
base_columns = [
    age_buckets,
]

model_imp = tf.estimator.LinearClassifier(
    model_dir="ongoing/train3",

feature_columns=categorical_features+base_columns+education_x_occupation+age_buckets_x_education_x_occupation)
```

SALIDA

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train3',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1823021be0>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

```

FEATURES_imp = ['age', 'workclass', 'education', 'education_num',
'marital',
          'occupation', 'relationship', 'race', 'sex',
'native_country', 'new']

def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in
FEATURES_imp}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)

```

Está listo para estimar el nuevo modelo y ver si mejora la precisión.

```

model_imp.train(input_fn=get_input_fn(df_train_new,
                                      num_epochs=None,
                                      n_batch = 128,
                                      shuffle=False),
                      steps=1000)

```

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
ongoing/train3/model.ckpt.
INFO:tensorflow:loss = 88.722855, step = 1
INFO:tensorflow:global_step/sec: 94.969
INFO:tensorflow:loss = 50.334488, step = 101 (1.054 sec)
INFO:tensorflow:global_step/sec: 242.342
INFO:tensorflow:loss = 56.153225, step = 201 (0.414 sec)
INFO:tensorflow:global_step/sec: 213.686
INFO:tensorflow:loss = 45.792007, step = 301 (0.470 sec)
INFO:tensorflow:global_step/sec: 174.084
INFO:tensorflow:loss = 37.485672, step = 401 (0.572 sec)
INFO:tensorflow:global_step/sec: 191.78
INFO:tensorflow:loss = 56.48449, step = 501 (0.524 sec)
INFO:tensorflow:global_step/sec: 163.436

```

```
INFO:tensorflow:loss = 32.528934, step = 601 (0.612 sec)
INFO:tensorflow:global_step/sec: 164.347
INFO:tensorflow:loss = 37.438057, step = 701 (0.607 sec)
INFO:tensorflow:global_step/sec: 154.274
INFO:tensorflow:loss = 61.1075, step = 801 (0.647 sec)
INFO:tensorflow:global_step/sec: 189.14
INFO:tensorflow:loss = 44.69645, step = 901 (0.531 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train3/model.ckpt.
INFO:tensorflow:Loss for final step: 44.18133.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x1823021cf8>
```

```
model_imp.evaluate(input_fn=get_input_fn(df_test_new,
                                         num_epochs=1,
                                         n_batch = 128,
                                         shuffle=False),
                     steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-02-08:28:52
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
ongoing/train3/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [100/1000]
INFO:tensorflow:Finished evaluation at 2018-06-02-08:28:54
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.8358209, accuracy_baseline = 0.76377374, auc = 0.88401634,
auc_precision_recall = 0.69599575, average_loss = 0.35122654,
global_step = 1000, label/mean = 0.23622628, loss = 44.67437,
precision = 0.68986726, prediction/mean = 0.23320661, recall =
0.55408216
```

```
{'accuracy': 0.8358209,
 'accuracy_baseline': 0.76377374,
```

```
'auc': 0.88401634,  
'auc_precision_recall': 0.69599575,  
'average_loss': 0.35122654,  
'global_step': 1000,  
'label/mean': 0.23622628,  
'loss': 44.67437,  
'precision': 0.68986726,  
'prediction/mean': 0.23320661,  
'recall': 0.55408216}
```

El nuevo nivel de precisión es 83.58 por ciento. Es un cuatro por ciento más alto que el modelo anterior.

Por último, puede agregar un término de regularización para evitar el exceso de ajuste.

Paso 5) Hiperparámetro: Lazo y cresta

Su modelo puede sufrir de **demasiado ajustado o Inapropiado**.

- Sobreajuste: el modelo no puede generalizar la predicción a nuevos datos
- Inadecuación: El modelo no puede capturar el patrón de los datos.
Es decir, regresión lineal cuando los datos no son lineales

Cuando un modelo tiene muchos parámetros y una cantidad relativamente baja de datos, conduce a predicciones deficientes. Imagine, un grupo solo tiene tres observaciones; el modelo calculará un peso para este grupo. El peso se usa para hacer una predicción; si las observaciones del conjunto de pruebas para este grupo en particular son completamente diferentes del conjunto de entrenamiento, entonces el modelo hará una predicción incorrecta. Durante la evaluación con el conjunto de entrenamiento, la precisión es buena, pero no buena con el conjunto de pruebas porque los pesos calculados no son los verdaderos para generalizar el patrón. En este caso, no hace una predicción razonable

sobre datos no vistos.

Para evitar el exceso de ajuste, la regularización le da las posibilidades de controlar dicha complejidad y hacerla más generalizable. Existen dos técnicas de regularización:

- L1: Lazo
- L2: Ridge

En TensorFlow, puede agregar estos dos hiperparámetros en el optimizador. Por ejemplo, cuanto mayor sea el hiperparámetro L2, el peso tiende a ser muy bajo y cercano a cero. La línea ajustada será muy plana, mientras que una L2 cercana a cero implica que los pesos están cerca de la regresión lineal regular.

Puede probar por sí mismo el valor diferente de los hiperparámetros y ver si puede aumentar el nivel de precisión.

Nota que si cambia el hiperparámetro, debe eliminar la carpeta en curso/train4 de lo contrario, el modelo comenzará con el modelo previamente entrenado.

Vamos a ver cómo es la precisión con el bombo

```
model_regu = tf.estimator.LinearClassifier(  
    model_dir="ongoing/train4",  
    feature_columns=categorical_features+base_columns+education_x_occupation+age_buckets_x_education_x_occupation,  
    optimizer=tf.train.FtrlOptimizer(  
        learning_rate=0.1,  
        l1_regularization_strength=0.9,  
        l2_regularization_strength=5))
```

SALIDA

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'ongoing/train4',
```

```
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1820d9c128>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

```
model_regu.train(input_fn=get_input_fn(df_train_new,
                                         num_epochs=None,
                                         n_batch = 128,
                                         shuffle=False),
                  steps=1000)
```

SALIDA

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
ongoing/train4/model.ckpt.
INFO:tensorflow:loss = 88.722855, step = 1
INFO:tensorflow:global_step/sec: 77.4165
INFO:tensorflow:loss = 50.38778, step = 101 (1.294 sec)
INFO:tensorflow:global_step/sec: 187.889
INFO:tensorflow:loss = 55.38014, step = 201 (0.535 sec)
INFO:tensorflow:global_step/sec: 201.895
INFO:tensorflow:loss = 46.806694, step = 301 (0.491 sec)
INFO:tensorflow:global_step/sec: 217.992
INFO:tensorflow:loss = 38.68271, step = 401 (0.460 sec)
INFO:tensorflow:global_step/sec: 193.676
INFO:tensorflow:loss = 56.99398, step = 501 (0.516 sec)
INFO:tensorflow:global_step/sec: 202.195
INFO:tensorflow:loss = 33.263622, step = 601 (0.497 sec)
INFO:tensorflow:global_step/sec: 216.756
INFO:tensorflow:loss = 37.7902, step = 701 (0.459 sec)
INFO:tensorflow:global_step/sec: 240.215
INFO:tensorflow:loss = 61.732605, step = 801 (0.416 sec)
```

```
INFO:tensorflow:global_step/sec: 220.336
INFO:tensorflow:loss = 46.938225, step = 901 (0.456 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
ongoing/train4/model.ckpt.
INFO:tensorflow:Loss for final step: 43.4942.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at
0x181ff39e48>
```

```
model_regu.evaluate(input_fn=get_input_fn(df_test_new,
                                         num_epochs=1,
                                         n_batch = 128,
                                         shuffle=False),
                     steps=1000)
```

SALIDA

```
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-06-02-08:29:07
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from
ongoing/train4/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [100/1000]
INFO:tensorflow:Finished evaluation at 2018-06-02-08:29:09
INFO:tensorflow:Saving dict for global step 1000: accuracy =
0.83833915, accuracy_baseline = 0.76377374, auc = 0.8869794,
auc_precision_recall = 0.7014905, average_loss = 0.34691378,
global_step = 1000, label/mean = 0.23622628, loss = 44.12581,
precision = 0.69720596, prediction/mean = 0.23662092, recall =
0.5579823

{'accuracy': 0.83833915,
```

```
'accuracy_baseline': 0.76377374,  
'auc': 0.8869794,  
'auc_precision_recall': 0.7014905,  
'average_loss': 0.34691378,  
'global_step': 1000,  
'label/mean': 0.23622628,  
'loss': 44.12581,  
'precision': 0.69720596,  
'prediction/mean': 0.23662092,  
'recall': 0.5579823}
```

Con este hiperparámetro, aumenta ligeramente las métricas de precisión. En el siguiente tutorial, aprenderá cómo mejorar un clasificador lineal utilizando un método kernel.

Resumen

Para entrenar un modelo, debe:

- Definir las funciones: Variables independientes: X
- Definir la etiqueta: Variable dependiente: y
- Construir un conjunto de tren/prueba
- Definir el peso inicial
- Definir la función de pérdida: MSE
- Optimizar el modelo: Descenso degradado
- Definir:
 - Tasa de aprendizaje
 - Número de época
 - Tamaño del lote
 - Número de clases

En este tutorial, aprendió a utilizar la API de alto nivel para un clasificador de regresión lineal. Debe definir:

1. Columnas de entidades. Si es continuo:
tf.feature_column.numeric_column (). Puede llenar una lista con

comprensión de lista de Python

2. El estimador: `tf.estimator.linearClassifier (feature_columns, model_dir, n_classes = 2)`
3. Una función para importar los datos, el tamaño del lote y la época: `input_fn ()`

Después de eso, usted está listo para entrenar, evaluar y hacer una predicción con `train ()`, `evaluate ()` y `predict ()`

Para mejorar el rendimiento del modelo, puede:

- Utilizar regresión polinómica
- Término de interacción: `tf.feature_column.crossed_column`
- Agregar parámetro de regularización

Capítulo 16: Métodos del núcleo

El propósito de este tutorial es hacer que un conjunto de datos separable linealmente. El tutorial se divide en dos partes:

1. Transformación de entidades
2. Entrena un clasificador de kernel con Tensorflow

En la primera parte, comprenderá la idea detrás de un clasificador de kernel mientras que en la segunda parte, verá cómo entrenar un clasificador de kernel con Tensorflow. Utilizará el conjunto de datos para adultos. El objetivo de este conjunto de datos es clasificar los ingresos por debajo y por encima de 50k, conociendo el comportamiento de cada hogar.

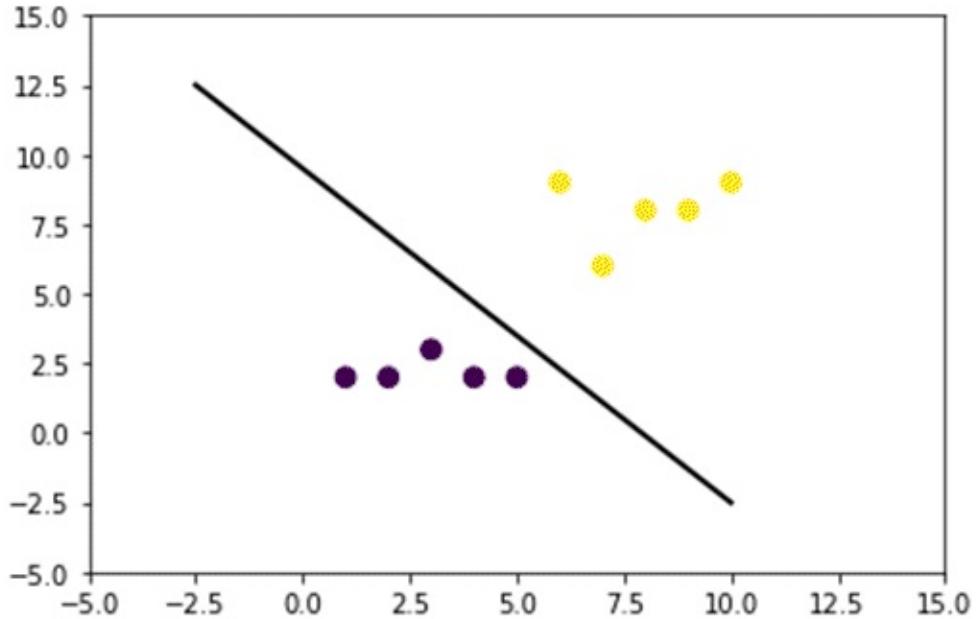
¿Por qué necesita Kernel Methods?

El objetivo de cada clasificador es predecir correctamente las clases. Para eso, el conjunto de datos debe ser separable. Mira la trama de abajo; es bastante simple ver que todos los puntos por encima de la línea negra pertenecen a la primera clase y los otros puntos a la segunda clase. Sin embargo, es extremadamente raro tener un conjunto de datos tan simple. En la mayoría de los casos, los datos no son separables. Le da a los clasificadores ingenuos como una regresión logística un momento difícil.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x_lin = np.array([1,2,3,4,5,6,7,8,9,10])
y_lin = np.array([2,2,3,2,2,9,6,8,8,9])
label_lin = np.array([0,0,0,0,0,1,1,1,1,1])

fig = plt.figure()
ax=fig.add_subplot(111)
plt.scatter(x_lin, y_lin, c=label_lin, s=60)
plt.plot([-2.5, 10], [12.5, -2.5], 'k-', lw=2)
ax.set_xlim([-5,15])
ax.set_ylim([-5,15])plt.show()
```

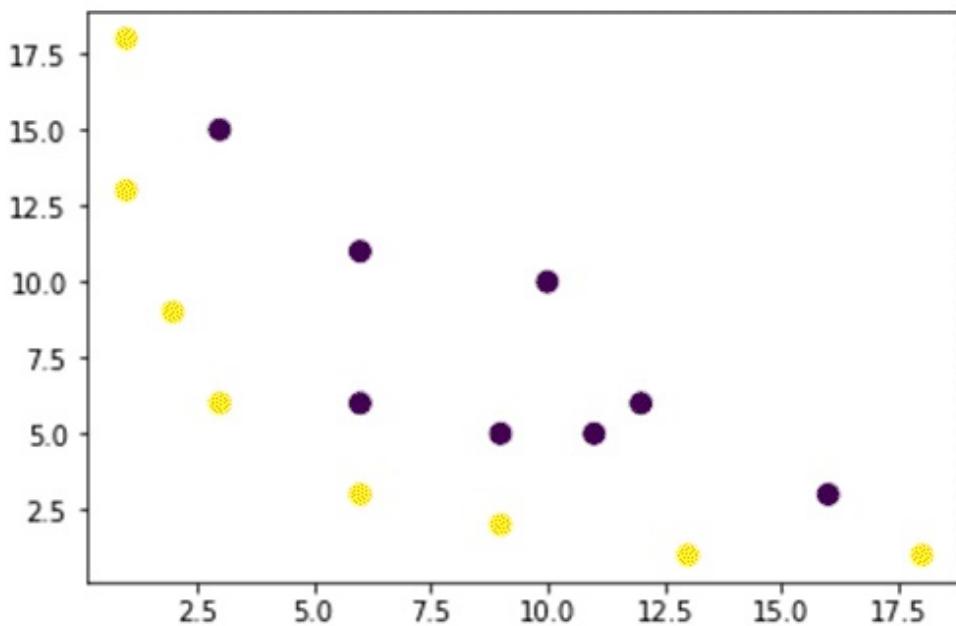


En la figura siguiente, trazamos un conjunto de datos que no es linealmente separable. Si trazamos una línea recta, la mayoría de los puntos no se clasificarán en la clase correcta.

Una forma de abordar este problema es tomar el dataset y transformar los datos en otro mapa de entidades. Significa que usará una función para transformar los datos en otro plan, que debería ser lineal.

```
x = np.array([1,1,2,3,3,6,6,6,9,9,10,11,12,13,16,18])
y = np.array([18,13,9,6,15,11,6,3,5,2,10,5,6,1,3,1])
label = np.array([1,1,1,1,0,0,0,1,0,1,0,0,0,1,0,1])
```

```
fig = plt.figure()
plt.scatter(x, y, c=label, s=60)
plt.show()
```



Los datos de la figura anterior se encuentran en un plan de dos dimensiones que no es separable. Puede intentar transformar estos datos en una dimensión de tres dimensiones, es decir, crear una figura con 3 ejes.

En nuestro ejemplo, aplicaremos un mapeo polinómico para llevar nuestros datos a una dimensión 3D. La fórmula para transformar los datos es la siguiente.

$$\phi(x, y) = (x^2, \sqrt{2}xy, y^2)$$

Defina una función en Python para crear los nuevos mapas de entidades

Puede usar numpy para codificar la fórmula anterior:

Fórmula	Código Numpy equivalente
x	<code>x[:,0]**</code>
y	<code>x[:,1]</code>
x^2	<code>x[:,0]**2</code>
$\sqrt{2}$	<code>np.sqrt(2)*</code>

xy	$x[:,0]*x[:,1]$
y ²	$x[:,1]**2$

```
### illustration purpose
def mapping(x, y):
    x = np.c_[(x, y)]
    if len(x) > 2:
        x_1 = x[:,0]**2
        x_2 = np.sqrt(2)*x[:,0]*x[:,1]
        x_3 = x[:,1]**2
    else:
        x_1 = x[0]**2
        x_2 = np.sqrt(2)*x[0]*x[1]
        x_3 = x[1]**2
    trans_x = np.array([x_1, x_2, x_3])
    return trans_x
```

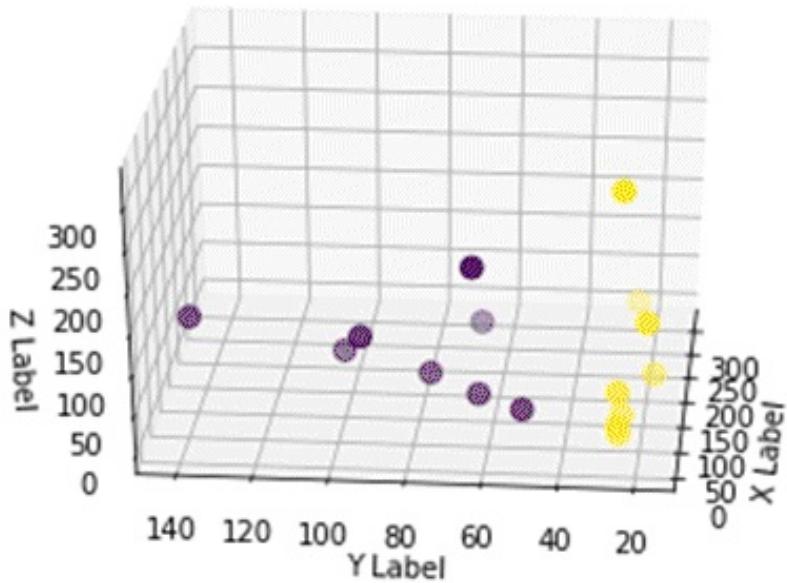
El nuevo mapeo debe ser con 3 dimensiones con 16 puntos

```
x_1 = mapping(x, y)
x_1.shape
```

```
(3, 16)
```

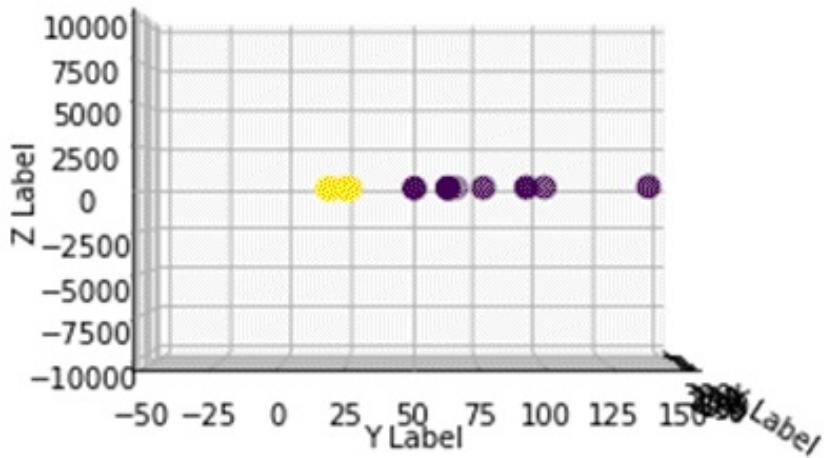
Vamos a hacer una nueva trama con 3 ejes, x, y y z respectivamente.

```
# plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_1[0], x_1[1], x_1[2], c=label, s=60)
ax.view_init(30, 185)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```



Vemos una mejora, pero si cambiamos la orientación de la trama, está claro que el conjunto de datos es ahora separable

```
# plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_1[0], x_1[1], x_1[1], c=label, s=60)
ax.view_init(0, -180)ax.set_xlim([150,-50])
ax.set_zlim([-10000,10000])
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')plt.show()
```



Para manipular un conjunto de datos grande y es posible que tenga que crear más de 2 dimensiones, se enfrentará a un gran problema utilizando el método anterior. De hecho, es necesario transformar todos los puntos de datos, lo que claramente no es sostenible. Le llevará años, y su computadora puede quedarse sin memoria.

La forma más común de superar este problema es usar un **kernel**.

¿Qué es un kernel en el aprendizaje automático?

La idea es utilizar un espacio de entidades de mayor dimensión para hacer que los datos se separen casi linealmente como se muestra en la figura anterior.

Hay muchos espacios dimensionales superiores para hacer que los puntos de datos sean separables. Por ejemplo, hemos demostrado que el mapeo polinómico es un gran comienzo.

También hemos demostrado que con muchos datos, esta transformación no es eficiente. En su lugar, puede utilizar una función de núcleo para modificar los datos sin cambiar a un nuevo plan de entidades.

La magia del núcleo es encontrar una función que evite todos los problemas que implica el cálculo de alta dimensión. El resultado de un núcleo es un escalar, o dicho de manera diferente estamos de vuelta al espacio unidimensional

Después de encontrar esta función, puede conectarla al clasificador lineal estándar.

Veamos un ejemplo para entender el concepto de Kernel. Tienes dos vectores, x_1 y x_2 . El objetivo es crear una dimensión superior mediante el uso de una asignación polinómica. La salida es igual al producto de punto del nuevo mapa de entidades. Desde el método anterior, debe:

1. Transformar x_1 y x_2 en una nueva dimensión
2. Calcular el producto de punto: común a todos los núcleos
3. Transformar x_1 y x_2 en una nueva dimensión

Puede utilizar la función creada anteriormente para calcular la dimensión superior.

```
## Kernel1
x1 = np.array([3,6])
x2 = np.array([10,10])

x_1 = mapping(x1, x2)
print(x_1)
```

Salida

```
[[ 9.          100.         ]
 [ 25.45584412 141.42135624]
 [ 36.          100.        ]]
```

Calcular el producto de puntos

Puede usar el punto objeto de numpy para calcular el producto de puntos entre el primer y el segundo vector almacenado en `x_1`.

```
print(np.dot(x_1[:,0], x_1[:,1]))
8100.0
```

La salida es 8100. Si ve el problema, debe almacenar en la memoria un nuevo mapa de entidades para calcular el producto de puntos. Si tiene un conjunto de datos con millones de registros, es computacionalmente ineficaz.

En su lugar, puede usar el método **núcleo polinómico** para calcular el producto de punto sin transformar el vector. Esta función calcula el producto punto de `x1` y `x2` como si estos dos vectores se hubieran transformado en la dimensión superior. Dicho de manera diferente, una función kernel calcula los resultados del producto punto desde otro espacio de características.

Puede escribir la función del núcleo polinomial en Python de la siguiente

manera.

```
def polynomial_kernel(x, y, p=2):
    return (np.dot(x, y)) ** p
```

Es el poder del producto punto de dos vectores. A continuación, devuelve el segundo grado del núcleo polinómico. La salida es igual al otro método. Esta es la magia del núcleo.

```
polynomial_kernel(x1, x2, p=2)
8100
```

Tipo de métodos de kernel

Hay un montón de granos diferentes disponibles. El más simple es el núcleo lineal. Esta función funciona bastante bien para la clasificación de texto. El otro núcleo es:

- Núcleo polinómico
- Núcleo gaussiana

En el ejemplo con TensorFlow, usaremos el Fourier aleatorio.

TensorFlow tiene un estimador de compilación para calcular el nuevo espacio de entidades. Esta función es una aproximación de la función kernel gaussiana.

$$e^{\frac{-\|x - y\|^2}{2\sigma^2}}$$

Esta función calcula la similitud entre los puntos de datos en un espacio dimensional mucho más alto.

Clasificador de Kernel Gaussian con TensorFlow

El objetivo del algoritmo es clasificar a los hogares que ganan más o menos de 50k.

Evaluará una regresión logística para tener un modelo de referencia. Después de eso, entrenará un clasificador Kernel para ver si puede obtener mejores resultados.

Utilice las siguientes variables del conjunto de datos para adultos:

- edad
- clase obrera
- Fnlwgt
- educación
- núm_de_educación
- conyugal
- ocupación
- relación
- raza
- relaciones sexuales
- ganancia_capital
- capital_pérdida
- horas_semana
- país_nativo
- etiqueta

Procederá de la siguiente manera antes de entrenar y evaluar el modelo:

- Paso 1) Importar las bibliotecas

- Paso 2) Importar los datos
- Paso 3) Preparar los datos
- Paso 4) Construir el input_fn
- Paso 5) Construir el modelo logístico: Modelo de línea base
- Paso 6) Evaluar el modelo
- Paso 7) Construir el clasificador de kernel
- Paso 8) Evaluar el clasificador de kernel

Paso 1) Importar las bibliotecas

Para importar y entrenar el modelo, debe importar tensorflow, pandas y numpy

```
#import numpy as np
from sklearn.model_selection
import train_test_split
import tensorflow as tf
import pandas as pd
import numpy as np
```

Paso 2) Importar los datos

Usted descarga los datos del siguiente sitio web e importa como un marco de datos panda.

```
## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex', 'capital_gain', 'capital_loss', 'hours_week',
'native_country', 'label']
PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
PATH_test ="https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.test
"## Import
df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
df_test = pd.read_csv(PATH_test,skiprows = 1,
skipinitialspace=True, names = COLUMNS, index_col=False)
```

Ahora que el tren y el conjunto de pruebas están definidos, puede cambiar la etiqueta de columna de cadena a integer. tensorflow no acepta el valor de cadena para la etiqueta.

```
label = {'<=50K': 0, '>50K': 1}
df_train.label = [label[item] for item in df_train.label]
label_t = {'<=50K.': 0, '>50K.': 1}
df_test.label = [label_t[item] for item in df_test.label]
df_train.shape

(32561, 15)
```

Paso 3) Preparar los datos

El dataset contiene entidades continuas y categóricas. Una buena práctica es estandarizar los valores de las variables continuas. Puede utilizar la función StandardScaler de sci-kit learn. También se crea una función definida por el usuario para facilitar la conversión del tren y el conjunto de pruebas. Tenga en cuenta que, concatenar las variables continuas y categóricas a un dataset común y la matriz debe ser del tipo: float32

```
COLUMNS_INT = ['age', 'fnlwgt', 'education_num', 'capital_gain',
'capital_loss', 'hours_week']
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
'relationship', 'race', 'sex', 'native_country']
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing

def prep_data_str(df):
    scaler = StandardScaler()
    le = preprocessing.LabelEncoder()
    df_toscale = df[COLUMNS_INT]
    df_scaled = scaler.fit_transform(df_toscale.astype(np.float64))
    X_1 = df[CATE_FEATURES].apply(le.fit_transform)
    y = df['label'].astype(np.int32)
    X_conc = np.c_[df_scaled, X_1].astype(np.float32)
    return X_conc, y
```

La función del transformador está lista, puede convertir el dataset y crear la función input_fn.

```
X_train, y_train = prep_data_str(df_train)
X_test, y_test = prep_data_str(df_test)
print(X_train.shape)
(32561, 14)
```

En el siguiente paso, entrenará una regresión logística. Le dará una precisión de línea de base. El objetivo es superar la línea de base con un algoritmo diferente, a saber, un clasificador Kernel.

Paso 4) Construir el modelo logístico: Modelo de línea base

La columna de entidad se construye con el objeto `real_valued_column`. Se asegurará de que todas las variables sean datos numéricos densos.

```
feat_column = tf.contrib.layers.real_valued_column('features',
dimension=14)
```

El estimador se define mediante TensorFlow Estimator, se indica a las columnas de entidades y dónde guardar el gráfico.

```
estimator = tf.estimator.LinearClassifier(feature_columns=
[feat_column],
                                         n_classes=2,
                                         model_dir = "kernel_log"
                                         )
```

```
INFO:tensorflow:Using default config.INFO:tensorflow:Using config:
{'_model_dir': 'kernel_log', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None,
'_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a2003f780>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

Entrenará la regresión lógica usando mini-lotes de tamaño 200.

```
# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"features": X_train},
    y=y_train,
    batch_size=200,
    num_epochs=None,
    shuffle=True)
```

Puede entrenar el modelo con 1.000 iteración

```
estimator.train(input_fn=train_input_fn, steps=1000)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
kernel_log/model.ckpt.
INFO:tensorflow:loss = 138.62949, step = 1
INFO:tensorflow:global_step/sec: 324.16
INFO:tensorflow:loss = 87.16762, step = 101 (0.310 sec)
INFO:tensorflow:global_step/sec: 267.092
INFO:tensorflow:loss = 71.53657, step = 201 (0.376 sec)
INFO:tensorflow:global_step/sec: 292.679
INFO:tensorflow:loss = 69.56703, step = 301 (0.340 sec)
INFO:tensorflow:global_step/sec: 225.582
INFO:tensorflow:loss = 74.615875, step = 401 (0.445 sec)
INFO:tensorflow:global_step/sec: 209.975
INFO:tensorflow:loss = 76.49044, step = 501 (0.475 sec)
INFO:tensorflow:global_step/sec: 241.648
INFO:tensorflow:loss = 66.38373, step = 601 (0.419 sec)
INFO:tensorflow:global_step/sec: 305.193
INFO:tensorflow:loss = 87.93341, step = 701 (0.327 sec)
INFO:tensorflow:global_step/sec: 396.295
INFO:tensorflow:loss = 76.61518, step = 801 (0.249 sec)
INFO:tensorflow:global_step/sec: 359.857
INFO:tensorflow:loss = 78.54885, step = 901 (0.277 sec)
INFO:tensorflow:Saving checkpoints for 1000 into
kernel_log/model.ckpt.
INFO:tensorflow:Loss for final step: 67.79706.
```

```
<tensorflow.python.estimator.canned.linear.LinearClassifier at  
0x1a1fa3cbe0>
```

Paso 6) Evaluar el modelo

Defina el estimador numpy para evaluar el modelo. Utilice el conjunto de datos completo para la evaluación

```
# Evaluation  
test_input_fn = tf.estimator.inputs.numpy_input_fn(  
    x={"features": X_test},  
    y=y_test,  
    batch_size=16281,  
    num_epochs=1,  
    shuffle=False)  
estimator.evaluate(input_fn=test_input_fn, steps=1)
```

```
INFO:tensorflow:Calling model_fn.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect  
PR-AUCs; please switch to "careful_interpolation" instead.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-07-12-15:58:22  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from kernel_log/model.ckpt-  
1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Evaluation [1/1]  
INFO:tensorflow:Finished evaluation at 2018-07-12-15:58:23  
INFO:tensorflow:Saving dict for global step 1000: accuracy =  
0.82353663, accuracy_baseline = 0.76377374, auc = 0.84898686,  
auc_precision_recall = 0.67214864, average_loss = 0.3877216,  
global_step = 1000, label/mean = 0.23622628, loss = 6312.495,  
precision = 0.7362797, prediction/mean = 0.21208474, recall =  
0.39417577
```

```
{'accuracy': 0.82353663,  
'accuracy_baseline': 0.76377374,  
'auc': 0.84898686,  
'auc_precision_recall': 0.67214864,  
'average_loss': 0.3877216,
```

```
'global_step': 1000,  
'label/mean': 0.23622628,  
'loss': 6312.495,  
'precision': 0.7362797,  
'prediction/mean': 0.21208474,  
'recall': 0.39417577}
```

Tiene una precisión del 82 por ciento. En la siguiente sección, intentarás vencer al clasificador logístico con un clasificador Kernel

Paso 7) Construir el clasificador de kernel

El estimador del núcleo no es tan diferente del clasificador lineal tradicional, al menos en términos de construcción. La idea detrás es usar el poder del kernel explícito con el clasificador lineal.

Necesita dos estimadores predefinidos disponibles en TensorFlow para entrenar el Kernel Classifier:

- RandomFourierFeatureMapper
- KernelLinearClassifier

Aprendió en la primera sección que necesita transformar la dimensión baja en una dimensión alta usando una función kernel. Más precisamente, usará el Fourier aleatorio, que es una aproximación de la función gaussiana. Afortunadamente, Tensorflow tiene la función en su biblioteca: RandomFourierFeatureMapper. El modelo se puede entrenar utilizando el estimador KernelLinearClassifier.

Para construir el modelo, siga estos pasos:

1. Establecer la función Kernel de dimensión alta
2. Establecer el hiperparámetro L2
3. Construir el modelo
4. Entrenar el modelo
5. Evaluar el modelo

Paso A) Establecer la función Kernel de alta dimensión

El dataset actual contiene 14 entidades que transformará en una nueva dimensión alta del vector de 5.000 dimensiones. Utilice las entidades aleatorias de Fourier para lograr la transformación. Si recuerda la fórmula Gaussian Kernel, observa que existe el parámetro de desviación estándar a definir. Este parámetro controla la medida de similitud empleada durante la clasificación.

Puede ajustar todos los parámetros en RandomFourierFeatureMapper con:

- input_dim = 14
- output_dim= 5000
- stddev=4

```
### Prep Kernel
kernel_mapper =
tf.contrib.kernel_methods.RandomFourierFeatureMapper(input_dim=14,
output_dim=5000, stddev=4, name='rffm')
```

Necesita construir el mapeador del núcleo utilizando las columnas de entidades creadas antes: feat_column

```
### Map Kernel
kernel_mappers = {feat_column: [kernel_mapper]}
```

Paso B) Establecer el hiperparámetro L2

Para evitar un ajuste excesivo, penaliza la función de pérdida con el regularizador L2. Establezca el hiperparámetro L2 en 0.1 y la tasa de aprendizaje en 5

```
optimizer = tf.train.FtrlOptimizer(learning_rate=5,
l2_regularization_strength=0.1)
```

Paso C) Construir el modelo

El siguiente paso es similar a la clasificación lineal. Se utiliza el estimador integrado KernelLinearClassifier. Tenga en cuenta que se agrega el asignador del núcleo definido anteriormente y se cambia el directorio del modelo.

```
### Prep estimator
estimator_kernel =
tf.contrib.kernel_methods.KernelLinearClassifier(
    n_classes=2,
    optimizer=optimizer,
    kernel_mappers=kernel_mappers,
    model_dir="kernel_train")
```

```
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/kernel_methods/python/kernel_estimators
.py:305: multi_class_head (from
tensorflow.contrib.learn.python.learn.estimators.head) is
deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.contrib.estimator.*_head.
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/learn/python/learn/estimators/estimator
.py:1179: BaseEstimator.__init__ (from
tensorflow.contrib.learn.python.learn.estimators.estimator) is
deprecated and will be removed in a future version.
Instructions for updating:
Please replace uses of any Estimator from tf.contrib.learn with an
Estimator from tf.estimator.*
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-
tf/lib/python3.6/site-
packages/tensorflow/contrib/learn/python/learn/estimators/estimator
.py:427: RunConfig.__init__ (from
tensorflow.contrib.learn.python.learn.estimators.run_config) is
deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use
tf.estimator.RunConfig instead.
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_task_type': None, '_task_id': 0,
'_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec
object at 0x1a200ae550>, '_master': '', '_num_ps_replicas': 0,
```

```
'_num_worker_replicas': 0, '_environment': 'local', '_is_chief': True, '_evaluation_master': '', '_train_distribute': None, '_tf_config': gpu_options { per_process_gpu_memory_fraction: 1.0 } , '_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_secs': 600, '_log_step_count_steps': 100, '_session_config': None, '_save_checkpoints_steps': None, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_model_dir': 'kernel_train'}
```

Paso D) Entrenar el modelo

Ahora que el clasificador Kernel está construido, ya está listo para entrenarlo. Elija iterar 2000 veces el modelo

```
### estimate
estimator_kernel.fit(input_fn=train_input_fn, steps=2000)

WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:From /Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-packages/tensorflow/contrib/learn/python/learn/estimators/head.py:678: ModelFnOps.__new__ (from tensorflow.contrib.learn.python.learn.estimators.model_fn) is deprecated and will be removed in a future version.
Instructions for updating:
When switching to tf.estimator.Estimator, use tf.estimator.EstimatorSpec. You can use the `estimator_spec` method to create an equivalent one.
INFO:tensorflow>Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into kernel_train/model.ckpt.
INFO:tensorflow:loss = 0.6931474, step = 1
```

```
INFO:tensorflow:global_step/sec: 86.6365
INFO:tensorflow:loss = 0.39374447, step = 101 (1.155 sec)
INFO:tensorflow:global_step/sec: 80.1986
INFO:tensorflow:loss = 0.3797774, step = 201 (1.247 sec)
INFO:tensorflow:global_step/sec: 79.6376
INFO:tensorflow:loss = 0.3908726, step = 301 (1.256 sec)
INFO:tensorflow:global_step/sec: 95.8442
INFO:tensorflow:loss = 0.41890752, step = 401 (1.043 sec)
INFO:tensorflow:global_step/sec: 93.7799
INFO:tensorflow:loss = 0.35700393, step = 501 (1.066 sec)
INFO:tensorflow:global_step/sec: 94.7071
INFO:tensorflow:loss = 0.35535482, step = 601 (1.056 sec)
INFO:tensorflow:global_step/sec: 90.7402
INFO:tensorflow:loss = 0.3692882, step = 701 (1.102 sec)
INFO:tensorflow:global_step/sec: 94.4924
INFO:tensorflow:loss = 0.34746957, step = 801 (1.058 sec)
INFO:tensorflow:global_step/sec: 95.3472
INFO:tensorflow:loss = 0.33655524, step = 901 (1.049 sec)
INFO:tensorflow:global_step/sec: 97.2928
INFO:tensorflow:loss = 0.35966292, step = 1001 (1.028 sec)
INFO:tensorflow:global_step/sec: 85.6761
INFO:tensorflow:loss = 0.31254214, step = 1101 (1.167 sec)
INFO:tensorflow:global_step/sec: 91.4194
INFO:tensorflow:loss = 0.33247527, step = 1201 (1.094 sec)
INFO:tensorflow:global_step/sec: 82.5954
INFO:tensorflow:loss = 0.29305756, step = 1301 (1.211 sec)
INFO:tensorflow:global_step/sec: 89.8748
INFO:tensorflow:loss = 0.37943482, step = 1401 (1.113 sec)
INFO:tensorflow:global_step/sec: 76.9761
INFO:tensorflow:loss = 0.34204718, step = 1501 (1.300 sec)
INFO:tensorflow:global_step/sec: 73.7192
INFO:tensorflow:loss = 0.34614792, step = 1601 (1.356 sec)
INFO:tensorflow:global_step/sec: 83.0573
INFO:tensorflow:loss = 0.38911164, step = 1701 (1.204 sec)
INFO:tensorflow:global_step/sec: 71.7029
INFO:tensorflow:loss = 0.35255936, step = 1801 (1.394 sec)
INFO:tensorflow:global_step/sec: 73.2663
INFO:tensorflow:loss = 0.31130585, step = 1901 (1.365 sec)
INFO:tensorflow:Saving checkpoints for 2000 into
kernel_train/model.ckpt.
INFO:tensorflow:Loss for final step: 0.37795097.
```

```
KernelLinearClassifier(params={'head':
<tensorflow.contrib.learn.python.learn.estimators.head._BinaryLogisticHead object at 0x1a2054cd30>, 'feature_columns':
```

```
{_RealValuedColumn(column_name='features_MAPPED', dimension=5000,
default_value=None, dtype=tf.float32, normalizer=None)},
'optimizer': <tensorflow.python.training.ftrl.FtrlOptimizer object
at 0x1a200aec18>, 'kernel_mappers':
{_RealValuedColumn(column_name='features', dimension=14,
default_value=None, dtype=tf.float32, normalizer=None):
[<tensorflow.contrib.kernel_methods.python.mappers.random_fourier_
features.RandomFourierFeatureMapper object at 0x1a200ae400>]}})
```

Paso E) Evaluar el modelo

Por último, pero no menos importante, evalúa el rendimiento de su modelo. Deberías ser capaz de superar la regresión logística.

```
# Evaluate and report metrics.
eval_metrics = estimator_kernel.evaluate(input_fn=test_input_fn,
steps=1)
```

```
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Casting <dtype: 'int32'> labels to bool.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
WARNING:tensorflow:Trapezoidal rule is known to produce incorrect
PR-AUCs; please switch to "careful_interpolation" instead.
INFO:tensorflow:Starting evaluation at 2018-07-12-15:58:50
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from kernel_train/model.ckpt-
2000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/1]
INFO:tensorflow:Finished evaluation at 2018-07-12-15:58:51
INFO:tensorflow:Saving dict for global step 2000: accuracy =
0.83975184, accuracy/baseline_label_mean = 0.23622628,
accuracy/threshold_0.500000_mean = 0.83975184, auc = 0.8904007,
auc_precision_recall = 0.72722375, global_step = 2000,
labels/actual_label_mean = 0.23622628, labels/prediction_mean =
0.23786618, loss = 0.34277728,
precision/positive_threshold_0.500000_mean = 0.73001117,
recall/positive_threshold_0.500000_mean = 0.5104004
```

La precisión final es del 84%, es una mejora del 2% respecto a

la regresión logística. Existe una compensación entre la mejora de la precisión y el costo computacional. Debe pensar si 2% mejora vale la pena el tiempo consumido por el clasificador diferente y si tiene un impacto convincente en su negocio.

Resumen

Un núcleo es una gran herramienta para transformar datos no lineales en (casi) lineales. La deficiencia de este método es computacionalmente lento y costoso.

A continuación, puede encontrar el código más importante para entrenar a un clasificador de kernel

Establecer la función Kernel de dimensión alta

- input_dim = 14
- output_dim= 5000
- stddev=4

```
### Prep Kernel
kernel_mapper =
tf.contrib.kernel_methods.RandomFourierFeatureMapper(input_dim=14,
output_dim=5000, stddev=4, name='rffm')
```

Establecer el hiperparámetro L2

```
optimizer = tf.train.FtrlOptimizer(learning_rate=5,
l2_regularization_strength=0.1)
```

Construir el modelo

```
estimator_kernel =
tf.contrib.kernel_methods.KernelLinearClassifier(      n_classes=2,
optimizer=optimizer,
kernel_mappers=kernel_mappers,
model_dir="kernel_train")
```

Entrenar el modelo

```
estimator_kernel.fit(input_fn=train_input_fn, steps=2000)
```

Evaluar el modelo

```
eval_metrics = estimator_kernel.evaluate(input_fn=test_input_fn,  
steps=1)
```

Capítulo 17: ANN de TensorFlow (Red Neural Artificial)

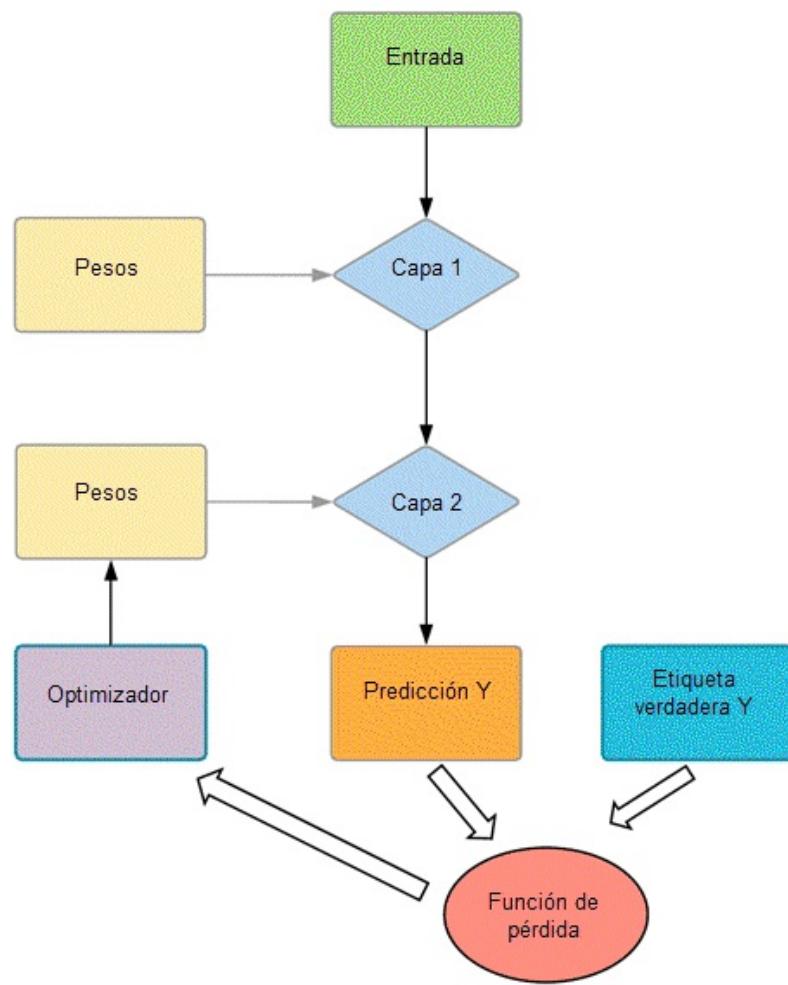
¿Qué es la red neuronal artificial?

Una Red Neural Artificial (ANN) se compone de cuatro objetos principales:

- Capas: todo el aprendizaje ocurre en las capas. Hay 3 capas 1) Entrada 2) Oculta y 3) Salida
- entidad y etiqueta: datos de entrada a la red (entidades) y salida de la red (etiquetas)
- función de pérdida: métrica utilizada para estimar el rendimiento de la fase de aprendizaje
- optimizador: Mejorar el aprendizaje actualizando los conocimientos en la red

Una red neuronal tomará los datos de entrada y los introducirá en un conjunto de capas. La red necesita evaluar su rendimiento con una función de pérdida. La función de pérdida da a la red una idea del camino que necesita tomar antes de dominar el conocimiento. La red necesita mejorar sus conocimientos con la ayuda de un optimizador.

Si echa un vistazo a la figura a continuación, comprenderá el mecanismo subyacente.



El programa toma algunos valores de entrada y los empuja en dos capas completamente conectadas. Imagina que tienes un problema matemático, lo primero que haces es leer el capítulo correspondiente para resolver el problema. Aplicas tus nuevos conocimientos para resolver el problema. Hay una gran probabilidad de que no anote muy bien. Es lo mismo para una red. La primera vez que vea los datos y haga una predicción, no coincidirá perfectamente con los datos reales.

Para mejorar sus conocimientos, la red utiliza un optimizador. En nuestra analogía, un optimizador puede ser considerado como releer el capítulo. Usted gana nuevas percepciones/lecciones leyendo de nuevo. Del mismo modo, la red utiliza el optimizador, actualiza sus conocimientos y prueba

sus nuevos conocimientos para comprobar cuánto aún necesita aprender. El programa repetirá este paso hasta que haga el error más bajo posible.

En nuestra analogía de problemas matemáticos, significa que lee el capítulo del libro de texto muchas veces hasta que entienda a fondo el contenido del curso. Incluso después de leer varias veces, si sigues cometiendo un error, significa que has alcanzado la capacidad de conocimiento con el material actual. Es necesario utilizar un libro de texto diferente o probar un método diferente para mejorar su puntuación. Para una red neuronal, es el mismo proceso. Si el error está lejos del 100%, pero la curva es plana, significa con la arquitectura actual; no puede aprender nada más. La red tiene que optimizarse mejor para mejorar el conocimiento.

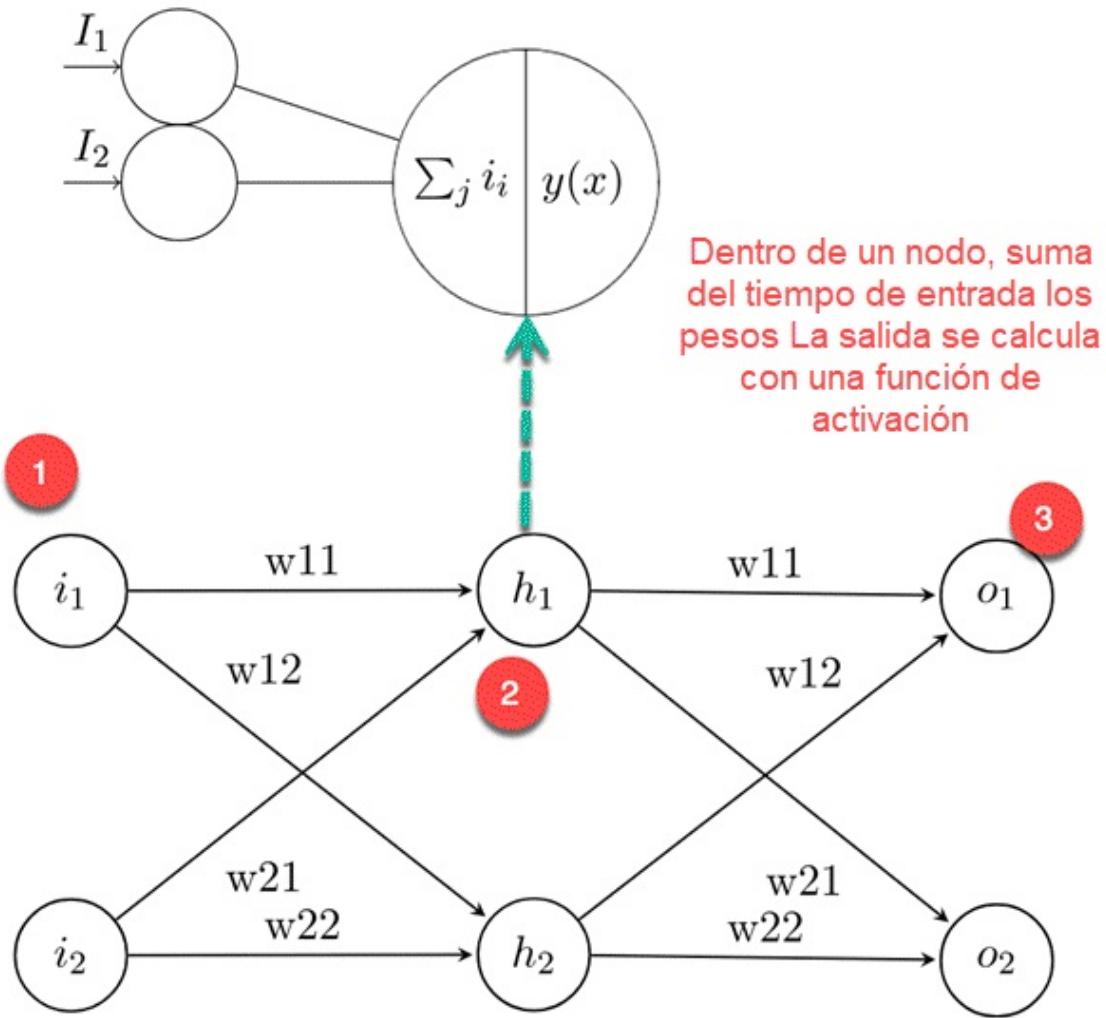
Arquitectura de red neuronal

Capas

Una capa es donde se lleva a cabo todo el aprendizaje. Dentro de una capa, hay una cantidad infinita de pesos (neuronas). Una red neuronal típica suele ser procesada por capas densamente conectadas (también llamadas capas completamente conectadas). Significa que todas las entradas están conectadas a la salida.

Una red neuronal típica toma un vector de entrada y un escalar que contiene las etiquetas. La configuración más cómoda es una clasificación binaria con solo dos clases: 0 y 1.

La red toma una entrada, la envía a todos los nodos conectados y calcula la señal con un **activación** función.



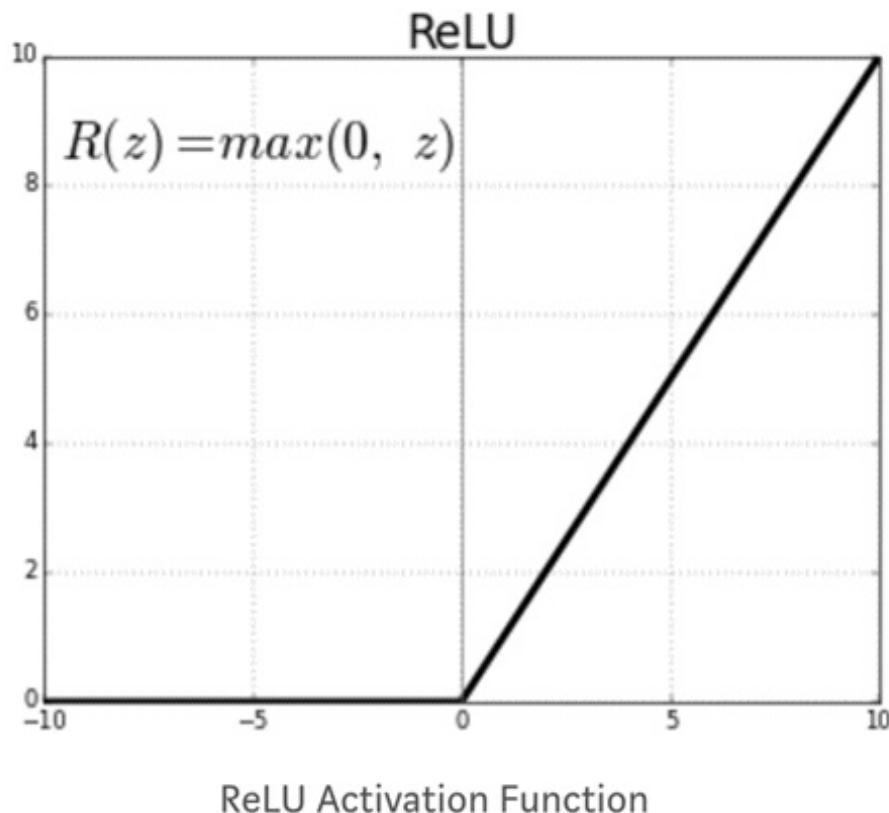
La figura anterior traza esta idea. La primera capa es los valores de entrada de la segunda capa, llamada capa oculta, recibe la entrada ponderada de la capa anterior

1. El primer nodo son los valores de entrada
2. La neurona se descompone en la parte de entrada y la función de activación. La parte izquierda recibe toda la entrada de la capa anterior. La parte derecha es la suma de la entrada pasa a una función de activación.
3. Valor de salida calculado a partir de las capas ocultas y utilizado para hacer una predicción. Para la clasificación, es igual al número de

clase. Para la regresión, sólo se predice un valor.

Función de activación

La función de activación de un nodo define la salida dada un conjunto de entradas. Necesita una función de activación para permitir que la red aprenda patrón no lineal. Una función de activación común es un **Relu, unidad lineal rectificada**. La función da un cero para todos los valores negativos.



Las otras funciones de activación son:

- Lineal a piezas
- Sigmoide
- Tanh
- Relu con fugas

La decisión crítica a tomar al construir una red neuronal es:

- ¿Cuántas capas en la red neuronal
- Quantas unidades ocultas para cada camada

La red neuronal con muchas capas y unidades ocultas puede aprender una representación compleja de los datos, pero hace que el cálculo de la red sea muy costoso.

Función de pérdida

Después de definir las capas ocultas y la función de activación, debe especificar la función de pérdida y el optimizador.

Para la clasificación binaria, es una práctica común usar una función de pérdida de entropía cruzada binaria. En la regresión lineal, se utiliza el error cuadrado medio.

La función de pérdida es una métrica importante para estimar el rendimiento del optimizador. Durante el entrenamiento, esta métrica se minimizará. Debe seleccionar esta cantidad cuidadosamente dependiendo del tipo de problema que esté tratando.

Optimizador

La función de pérdida es una medida del rendimiento del modelo. El optimizador ayudará a mejorar los pesos de la red con el fin de disminuir la pérdida. Hay diferentes optimizadores disponibles, pero el más común es el Descenso de Gradiente Estocástico.

Los optimizadores convencionales son:

- Optimización del momento,

- Nesterov gradiente acelerado,
- Adagrad,
- Optimización de Adam

Limitaciones de la red neuronal

Sobreajustado

Un problema común con la compleja red neuronal son las dificultades para generalizar datos no vistos. Una red neuronal con muchos pesos puede identificar detalles específicos en el conjunto de trenes muy bien, pero a menudo conduce a un ajuste excesivo. Si los datos están desequilibrados dentro de los grupos (es decir, no hay suficientes datos disponibles en algunos grupos), la red aprenderá muy bien durante el entrenamiento, pero no tendrá la capacidad de generalizar dicho patrón a datos nunca vistos antes.

Existe una compensación en el aprendizaje automático entre la optimización y la generalización.

Optimizar un modelo requiere encontrar los mejores parámetros que minimicen la pérdida del conjunto de entrenamiento.

La generalización, sin embargo, indica cómo se comporta el modelo para los datos no vistos.

Para evitar que el modelo capture detalles específicos o patrones no deseados de los datos de entrenamiento, puede utilizar diferentes técnicas. El mejor método es tener un conjunto de datos equilibrado con suficiente cantidad de datos. El arte de reducir el exceso de ajuste se llama **regularización**. Vamos a revisar algunas técnicas convencionales.

Tamaño de la red

Una red neuronal con demasiadas capas y unidades ocultas es conocida por ser altamente sofisticada. Una forma directa de reducir la

complejidad del modelo es reducir su tamaño. No se recomienda definir el número de capas. Debe comenzar con una pequeña cantidad de capa y aumenta su tamaño hasta que encuentre el exceso de ajuste del modelo.

Regularización de peso

Una técnica estándar para evitar el ajuste excesivo es agregar restricciones a los pesos de la red. La restricción obliga al tamaño de la red a tomar sólo valores pequeños. La restricción se agrega a la función de pérdida del error. Hay dos tipos de regularización:

L1: Lazo: El coste es proporcional al valor absoluto de los coeficientes de peso

L2: Ridge: El costo es proporcional al cuadrado del valor de los coeficientes de peso

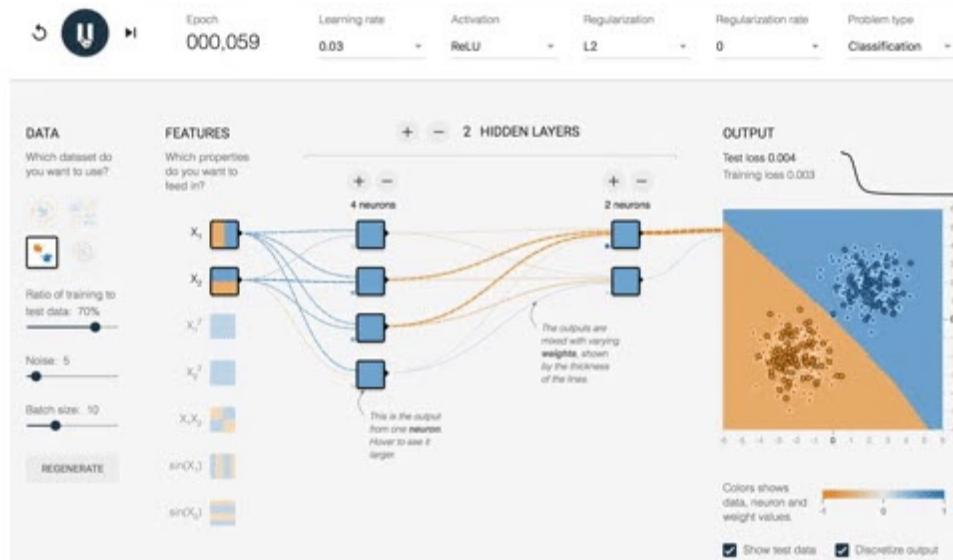
Estudiante

La deserción es una técnica extraña pero útil. Una red con abandono significa que algunos pesos se establecerán aleatoriamente en cero. Imagine que tiene una matriz de pesos $[0.1, 1.7, 0.7, -0.9]$. Si la red neuronal tiene una caída, se convertirá en $[0.1, 0, 0, -0.9]$ con o distribuido aleatoriamente. El parámetro que controla la deserción es la tasa de abandono. La tasa define el número de pesos que se deben establecer en ceros. Tener una tasa entre 0.2 y 0.5 es común.

Ejemplo de red neuronal en TensorFlow

Veamos en acción cómo funciona una red neuronal para un problema de clasificación típico. Hay dos entradas, x_1 y x_2 con un valor aleatorio. La salida es una clase binaria. El objetivo es clasificar la etiqueta en función de las dos características. Para llevar a cabo esta tarea, la arquitectura de la red neuronal se define de la siguiente manera:

- Dos capas ocultas
 - La primera capa tiene cuatro neuronas completamente conectadas
 - La segunda capa tiene dos neuronas totalmente conectadas
- La función de activación es un Relu
- Añadir una Regularización L2 con una tasa de aprendizaje de 0.003

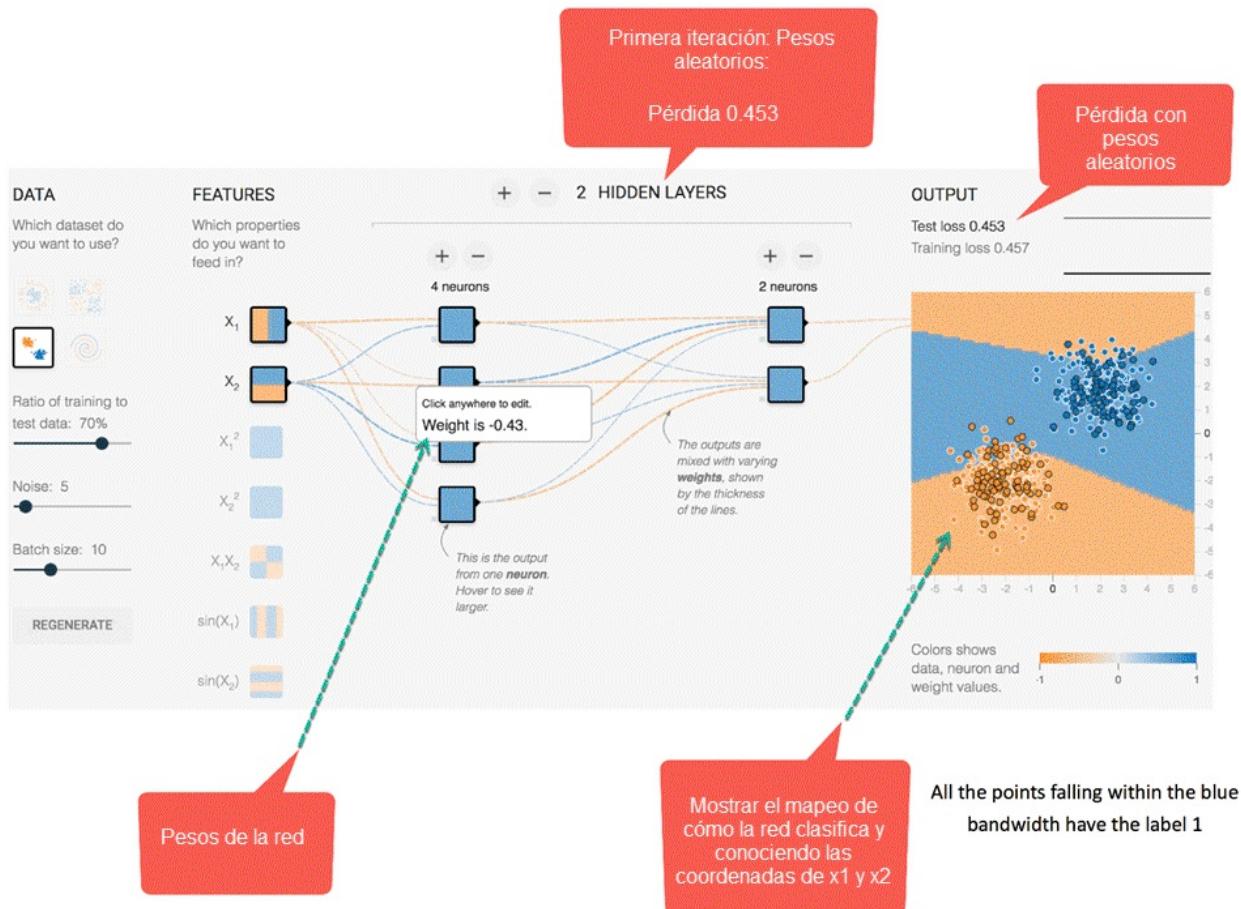


La red optimizará el peso durante 180 épocas con un tamaño de lote de 10. En el siguiente vídeo se puede ver cómo evolucionan los pesos y cómo

la red mejora la asignación de clasificación.

En primer lugar, la red asigna valores aleatorios a todos los pesos.

- Con los pesos aleatorios, es decir, sin optimización, la pérdida de salida es 0.453. La imagen a continuación representa la red con diferentes colores.
- En general, el color naranja representa valores negativos mientras que los colores azules muestran los valores positivos.
- Los puntos de datos tienen la misma representación; los azules son las etiquetas positivas y el naranja las etiquetas negativas.



Dentro de la segunda capa oculta, las líneas se colorean siguiendo el signo de los pesos. Las líneas anaranjadas asignan pesos negativos y la azul un peso positivo

Como puede ver, en el mapeo de salida, la red está cometiendo bastantes errores. Veamos cómo se comporta la red después de la optimización.

La siguiente imagen muestra los resultados de la red optimizada. En primer lugar, observa que la red ha aprendido con éxito cómo clasificar el punto de datos. Puede ver en la imagen anterior; el peso inicial fue -0.43 mientras que después de la optimización resulta en un peso de -0.95.



La idea puede generalizarse para redes con capas y neuronas más ocultas. Puedes jugar en el enlace.

Entrenar una red neuronal con TensorFlow

En esta parte del tutorial, aprenderá cómo entrenar una red neuronal con TensorFlow utilizando el estimador DNNClassifier de la API.

Usaremos el conjunto de datos MNIST para entrenar su primera red neuronal. El entrenamiento de una red neuronal con Tensorflow no es muy complicado. El paso de preprocesamiento se ve exactamente igual que en los tutoriales anteriores. Procederá de la siguiente manera:

- Paso 1: Importar los datos
- Paso 2: Transformar los datos
- Paso 3: Construir el tensor
- Paso 4: Construir el modelo
- Paso 5: Entrenar y evaluar el modelo
- Paso 6: Mejorar el modelo

Paso 1) Importar los datos

En primer lugar, debe importar la biblioteca necesaria. Puede importar el conjunto de datos MNIST utilizando scikit learn.

El conjunto de datos MNIST es el conjunto de datos comúnmente utilizado para probar nuevas técnicas o algoritmos. Este conjunto de datos es una colección de imagen de 28x28 píxeles con un dígito manuscrito de 0 a 9. Actualmente, el error más bajo en la prueba es de 0,27 por ciento con un comité de 7 redes neuronales convolucionales.

```
import numpy as np
import tensorflow as tf
np.random.seed(1337)
```

Puede descargar scikit learn temporalmente en esta dirección. Copie y pegue el dataset en una carpeta conveniente. Para importar los datos a python, puede usar fetch_mldata de scikit learn. Pegue la ruta del archivo dentro de fetch_mldata para recuperar los datos.

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata(
    '/Users/Thomas/Dropbox/Learning/Upwork/tuto_TF/data/ml_data/MNIST
original')
print(mnist.data.shape)
print(mnist.target.shape)
```

Después de eso, importa los datos y obtiene la forma de ambos datasets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(mnist.data,
mnist.target, test_size=0.2, random_state=42)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
batch_size = len(X_train)

print(X_train.shape, y_train.shape, y_test.shape )
```

Paso 2) Transforme los datos

En el tutorial anterior, aprendió que necesita transformar los datos para limitar el efecto de los valores atípicos. En este tutorial, transformará los datos utilizando el escalador min-max. La fórmula es:

$$(X - \min_X) / (\max_X - \min_X)$$

Scikit aprende ya tiene una función para eso: `minMaxScaler()`

```
## resclae
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Train
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
```

```
# test  
X_test_scaled = scaler.fit_transform(X_test.astype(np.float64))
```

Paso 3) Construir el tensor

Ahora está familiarizado con la forma de crear tensor en Tensorflow. Puede convertir el conjunto de trenes en una columna numérica.

```
feature_columns = [tf.feature_column.numeric_column('x',  
shape=X_train_scaled.shape[1:])]
```

Paso 4) Construir el modelo

La arquitectura de la red neuronal contiene 2 capas ocultas con 300 unidades para la primera capa y 100 unidades para la segunda. Utilizamos estos valores basados en nuestra propia experiencia. Puede ajustar estos valores y ver cómo afecta a la precisión de la red.

Para crear el modelo, utilice el estimador DNNClassifier. Puede agregar el número de capas a los argumentos feature_columns. Debe establecer el número de clases en 10, ya que hay diez clases en el conjunto de entrenamiento. Ya está familiarizado con la sintaxis del objeto estimador. Las columnas de características de argumentos, el número de clases y model_dir son exactamente los mismos que en el tutorial anterior. El nuevo argumento hidden_unit controla el número de capas y cuántos nodos se conectan a la red neuronal. En el siguiente código, hay dos capas ocultas con una primera que conecta 300 nodos y la segunda con 100 nodos.

Para crear el estimador, utilice tf.estimator.dnnClassifier con los siguientes parámetros:

- feature_columns: Define las columnas que se van a utilizar en la red
- hidden_units: Define el número de neuronas ocultas

- n_classes: Define el número de clases a predecir
- model_dir: Definir la ruta de TensorBoard

```
estimator = tf.estimator.DNNClassifier(
    feature_columns=feature_columns,
    hidden_units=[300, 100],
    n_classes=10,
    model_dir = '/train/DNN')
```

Paso 5) Entrenar y evaluar el modelo

Puede usar el método numpy para entrenar el modelo y evaluarlo

```
# Train the estimator
train_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train_scaled},
    y=y_train,
    batch_size=50,
    shuffle=False,
    num_epochs=None)
estimator.train(input_fn = train_input, steps=1000)
eval_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_test_scaled},
    y=y_test,
    shuffle=False,
    batch_size=X_test_scaled.shape[0],
    num_epochs=1)
estimator.evaluate(eval_input, steps=None)
```

Salida:

```
{'accuracy': 0.9637143,
'average_loss': 0.12014342,
'loss': 1682.0079,
'global_step': 1000}
```

La arquitectura actual conduce a una precisión en el conjunto de evaluación del 96 por ciento.

Paso 6) Mejorar el modelo

Puede intentar mejorar el modelo agregando parámetros de regularización.

Vamos a utilizar un optimizador Adam con una tasa de abandono de 0.3, L1 de X y L2 de y. En TensorFlow, usted puede controlar el optimizador usando el tren de objetos siguiendo el nombre del optimizador. TensorFlow es una API incorporada para el optimizador Adagrad Proximal.

Para agregar regularización a la red neuronal profunda, puede usar `tf.train.ProximalAdagradOptimizer` con el siguiente parámetro

- Tasa de aprendizaje: `learning_rate`
- Regularización L1: `l1_regularization_strength`
- Regularización L2: `l2_regularization_strength`

```
estimator_imp = tf.estimator.DNNClassifier(
    feature_columns=feature_columns,
    hidden_units=[300, 100],
    dropout=0.3,
    n_classes = 10,
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.01,
        l1_regularization_strength=0.01,
        l2_regularization_strength=0.01
    ),
    model_dir = '/train/DNN1')
estimator_imp.train(input_fn = train_input, steps=1000)
estimator_imp.evaluate(eval_input, steps=None)
```

Salida:

```
{'accuracy': 0.95057142,
'average_loss': 0.17318928,
'loss': 2424.6499,
'global_step': 2000}
```

Los valores elegidos para reducir la sobreconexión no mejoraron la

precisión del modelo. Su primer modelo tenía una precisión del 96%, mientras que el modelo con regularizador L2 tiene una precisión del 95%. Puede probar con diferentes valores y ver cómo afecta la precisión.

Resumen

En este tutorial, aprenderá a construir una red neuronal. Una red neuronal requiere:

- Número de capas ocultas
- Número de nodo completamente conectado
- Función de activación
- Optimizador
- Número de clases

En TensorFlow, puede entrenar una red neuronal para el problema de clasificación con:

- `TF.estimator.DNNClassifier`

El estimador requiere especificar:

- `feature_columns = feature_columns,`
- `unidades_ocultas = [300, 100]`
- `n_classes = 10`
- `dir_modelo`

Puede mejorar el modelo utilizando diferentes optimizadores. En este tutorial, aprendió a utilizar el optimizador Adam Grad con una tasa de aprendizaje y agregar un control para evitar el exceso de ajuste.

Capítulo 18: ConvNet (red neuronal convolucional): Clasificación de imágenes de TensorFlow

¿Qué es la red neuronal convolucional?

La red neuronal convolucional, también conocida como convnets o CNN, es un método bien conocido en aplicaciones de visión por computadora. Este tipo de arquitectura es dominante para reconocer objetos de una imagen o vídeo.

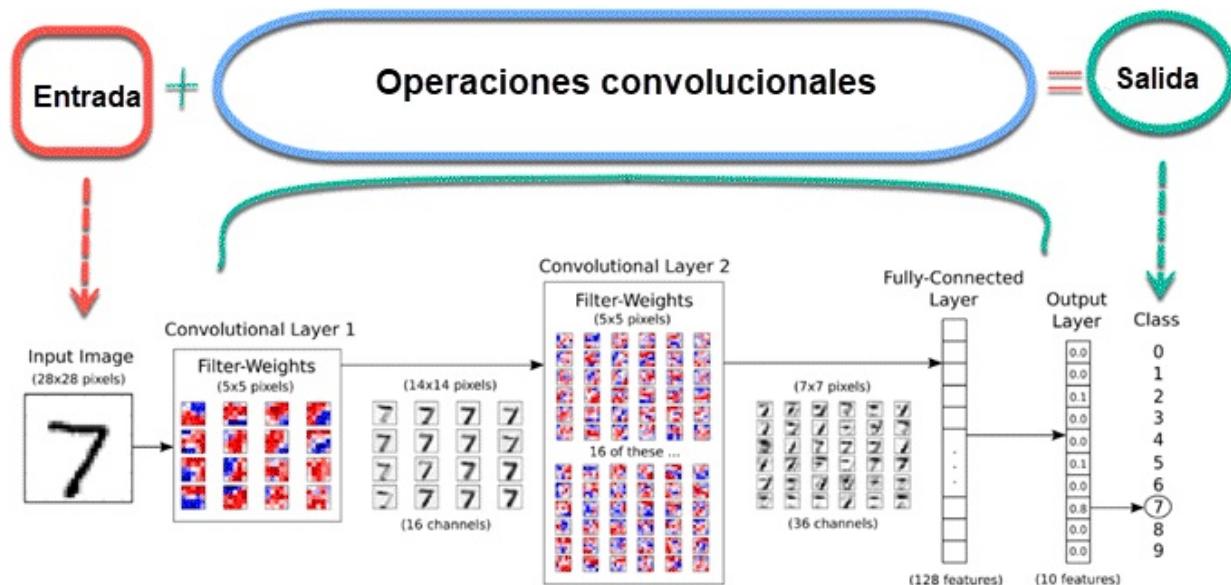
En este tutorial, aprenderá cómo construir una convnet y cómo utilizar TensorFlow para resolver el conjunto de datos escrito a mano.

Arquitectura de una red neuronal convolucional

Piense en Facebook hace unos años, después de subir una imagen a su perfil, se le pidió que agregue un nombre a la cara en la imagen manualmente. Hoy en día, Facebook usa convnet para etiquetar a tu amigo en la imagen automáticamente.

Una red neuronal convolucional no es muy difícil de entender. Una imagen de entrada se procesa durante la fase de convolución y posteriormente se le atribuye una etiqueta.

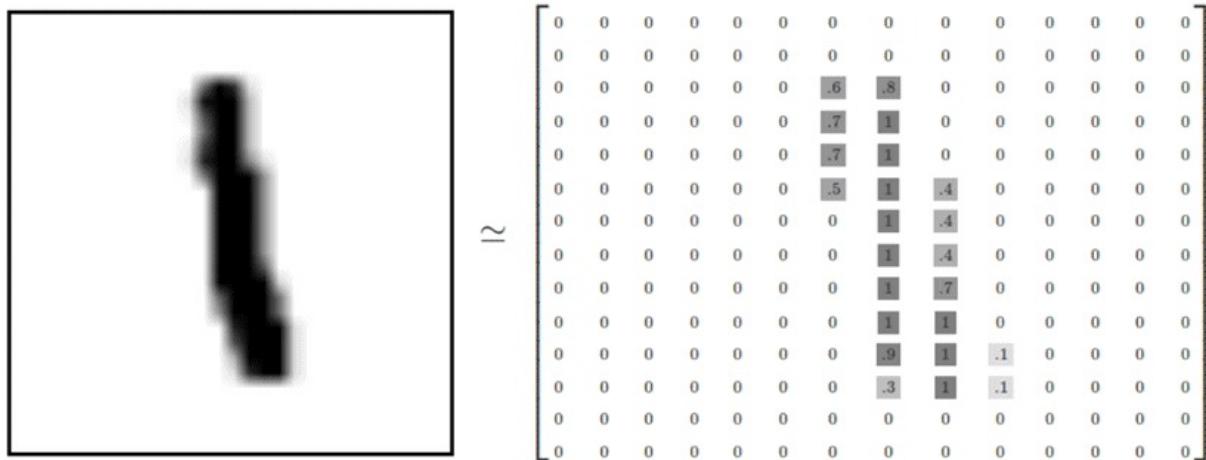
Una arquitectura convnet típica se puede resumir en la siguiente imagen. En primer lugar, se envía una imagen a la red; esto se llama imagen de entrada. Luego, la imagen de entrada pasa por un número infinito de pasos; esta es la parte convolucional de la red. Finalmente, la red neuronal puede predecir el dígito en la imagen.



Una imagen se compone de una matriz de píxeles con altura y ancho. Una

imagen en escala de grises sólo tiene un canal, mientras que la imagen en color tiene tres canales (cada uno para rojo, verde y azul). Un canal se apila uno sobre el otro. En este tutorial, utilizará una imagen en escala de grises con un solo canal. Cada píxel tiene un valor de 0 a 255 para reflejar la intensidad del color. Por ejemplo, un píxel igual a 0 mostrará un color blanca mientras que el píxel con un valor cercano a 255 será más oscuro.

Echemos un vistazo a una imagen almacenada en el dataset MNIST. La siguiente imagen muestra cómo representar la imagen de la izquierda en un formato de matriz. Tenga en cuenta que, la matriz original se ha estandarizado para estar entre 0 y 1. Para el color más oscuro, el valor de la matriz es aproximadamente 0,9, mientras que los píxeles blancos tienen un valor de 0.



Operación convolucional

El componente más crítico del modelo es la capa convolucional. Esta parte tiene como objetivo reducir el tamaño de la imagen para cálculos más rápidos de los pesos y mejorar su generalización.

Durante la parte convolucional, la red mantiene las características esenciales de la imagen y excluye el ruido irrelevante. Por ejemplo, el

modelo está aprendiendo a reconocer a un elefante de una imagen con una montaña en el fondo. Si utiliza una red neuronal tradicional, el modelo asignará un peso a todos los píxeles, incluidos los de la montaña, lo que no es esencial y puede inducir a error a la red.

En su lugar, una red neuronal convolucional utilizará una técnica matemática para extraer sólo los píxeles más relevantes. Esta operación matemática se llama convolución. Esta técnica permite a la red aprender entidades cada vez más complejas en cada capa. La convolución divide la matriz en trozos pequeños para aprender a los elementos más esenciales dentro de cada pieza.

Componentes de Convnets

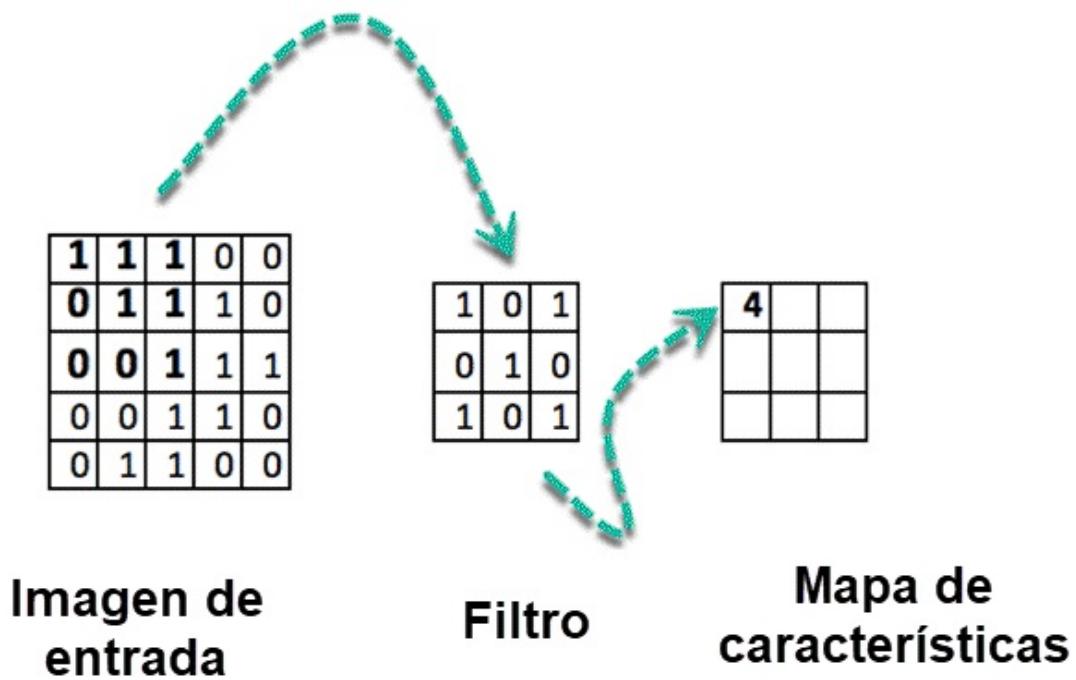
Hay cuatro componentes de un Convnets

1. Convolución
2. No linealidad (ReLU)
3. Agrupación o submuestreo
4. Clasificación (capa completamente conectada)
 - Convolución

El propósito de la convolución es extraer las características del objeto en la imagen localmente. Esto significa que la red aprenderá patrones específicos dentro de la imagen y será capaz de reconocerla en todas partes de la imagen.

La convolución es una multiplicación elemental. El concepto es fácil de entender. El ordenador escaneará una parte de la imagen, generalmente con una dimensión de 3x3 y la multiplica en un filtro. La salida de la multiplicación del elemento se denomina mapa de entidades. Este paso se repite hasta que se escanea toda la imagen. Tenga en cuenta que, después de la convolución, el tamaño de la imagen se reduce.

Convolución



A continuación, hay una URL para ver en acción cómo funciona la convolución.

1	1	1	0	0
0	1	1	1	0
0	0	1 _{×1}	1 _{×0}	1 _{×1}
0	0	1 _{×0}	1 _{×1}	0 _{×0}
0	1	1 _{×1}	0 _{×0}	0 _{×1}

Imagen

4	3	4
2	4	3
2	3	4

Función convolved

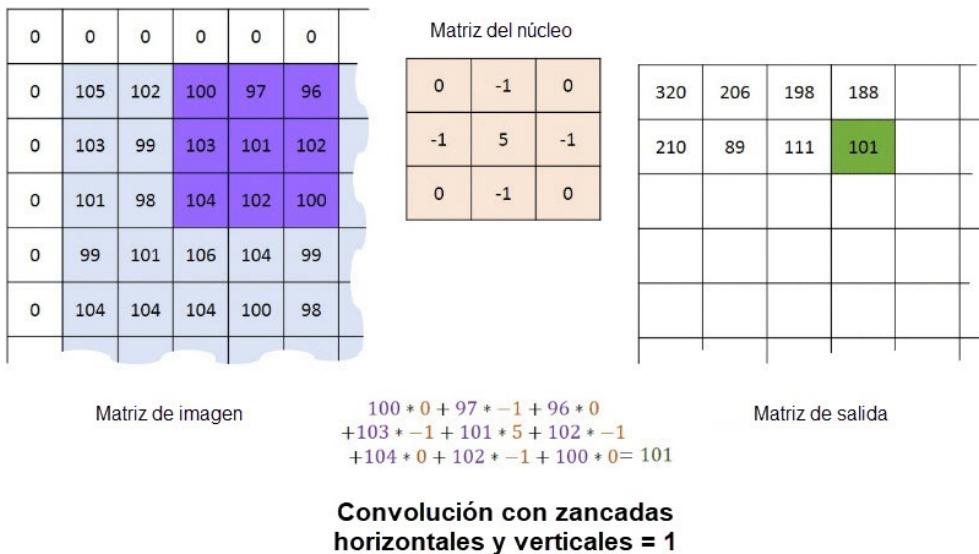
Hay numerosos canales disponibles. A continuación, enumeramos algunos de los canales. Puede ver que cada filtro tiene un propósito específico. Tenga en cuenta, en la imagen de abajo; el núcleo es un sinónimo del filtro.

Operación	Núcleo	Resultado de la imagen
Afilar	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Detección de bordes	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Aritmética detrás de la convolución

La fase convolucional aplicará el filtro en una pequeña matriz de píxeles dentro de la imagen. El filtro se moverá a lo largo de la imagen de entrada con una forma general de 3x3 o 5x5. Significa que la red deslizará estas

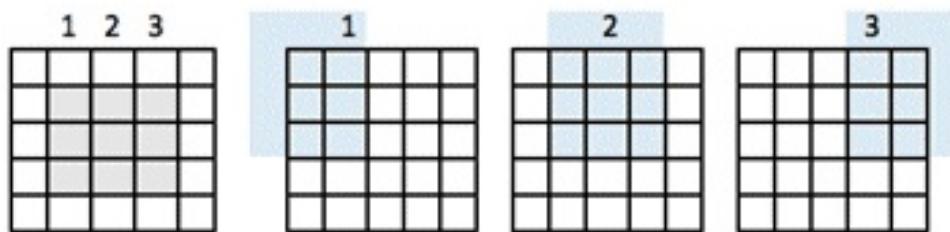
ventanas a través de toda la imagen de entrada y calculará la convolución. La siguiente imagen muestra cómo funciona la convolución. El tamaño del parche es 3x3, y la matriz de salida es el resultado de la operación de elemento entre la matriz de imagen y el filtro.



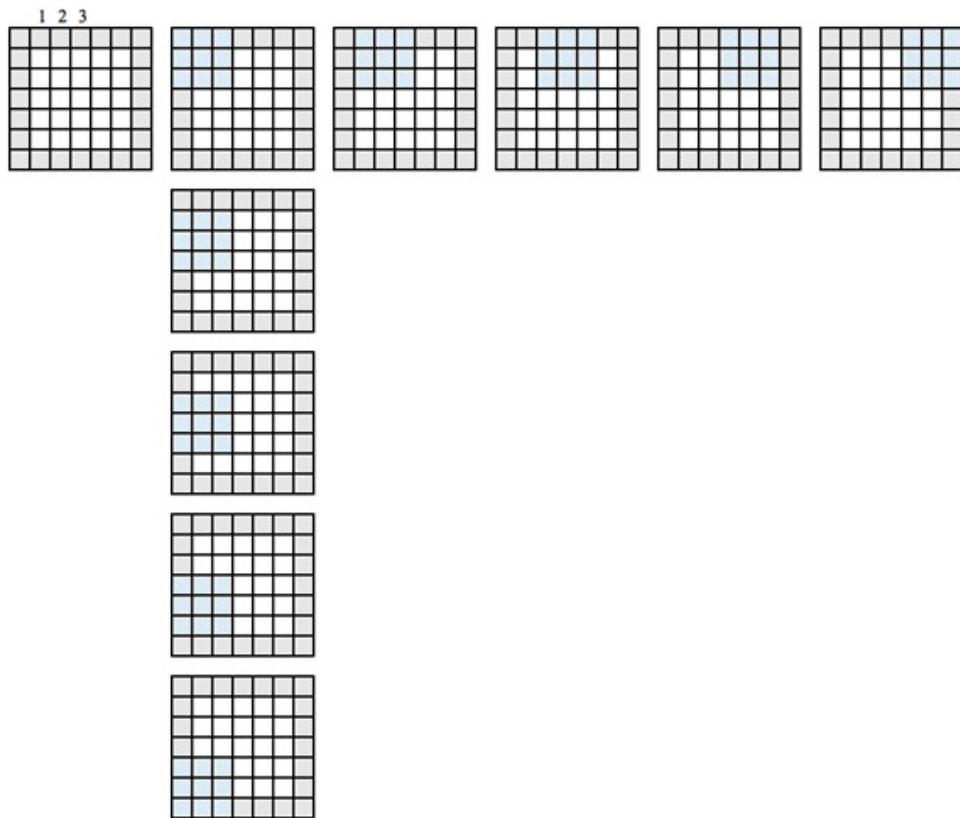
Observe que el ancho y la altura de la salida pueden ser diferentes de la anchura y la altura de la entrada. Sucede debido al efecto de borde.

Efecto Borde

La imagen tiene un mapa de características 5x5 y un filtro 3x3. Sólo hay una ventana en el centro donde el filtro puede mostrar una cuadrícula 3x3. El mapa de entidades de salida se reducirá en dos teselas junto con una dimensión 3x3.

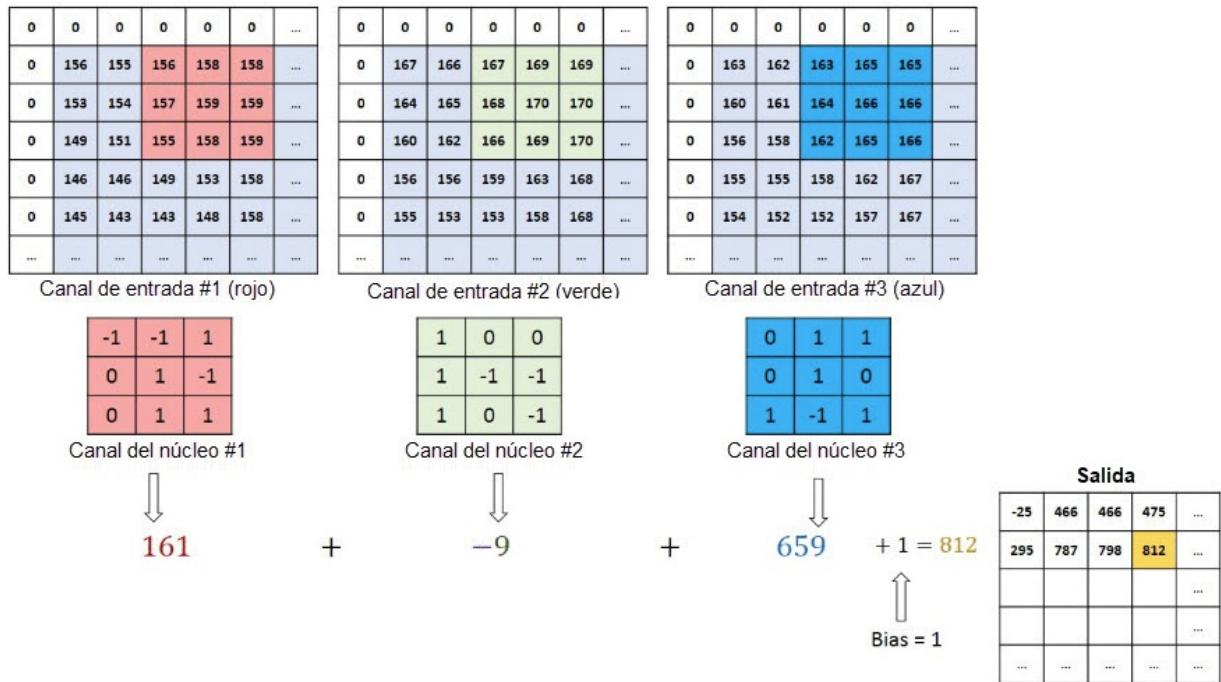


Para obtener la misma dimensión de salida que la dimensión de entrada, debe agregar relleno. El relleno consiste en agregar el número correcto de filas y columnas a cada lado de la matriz. Permitirá que la convolución se ajuste a cada tesela de entrada. En la imagen de abajo, la matriz de entrada/salida tiene la misma dimensión 5x5



Al definir la red, las entidades convolved se controlan mediante tres parámetros:

1. Profundidad: Define el número de filtros a aplicar durante la convolución. En el ejemplo anterior, vio una profundidad de 1, lo que significa que sólo se utiliza un filtro. En la mayoría de los casos, hay más de un filtro. La siguiente imagen muestra las operaciones realizadas en una situación con tres filtros



2. Stride: Define el número de “salto de píxel” entre dos rebanadas. Si la zancada es igual a 1, las ventanas se moverán con una extensión de píxel de uno. Si la zancada es igual a dos, las ventanas saltarán 2 píxeles. Si aumenta la zancada, tendrá mapas de entidades más pequeños.

Ejemplo de Stride 1

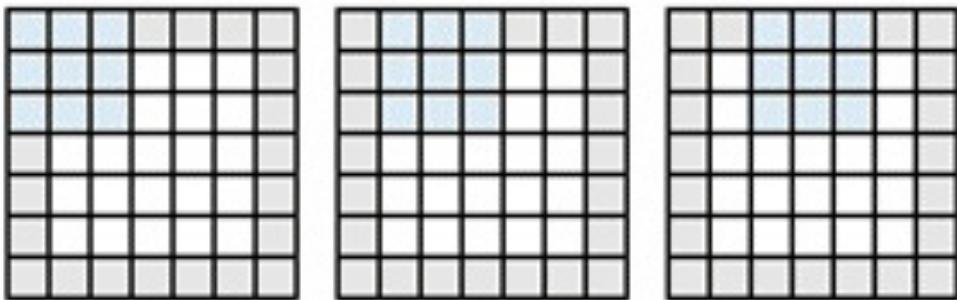
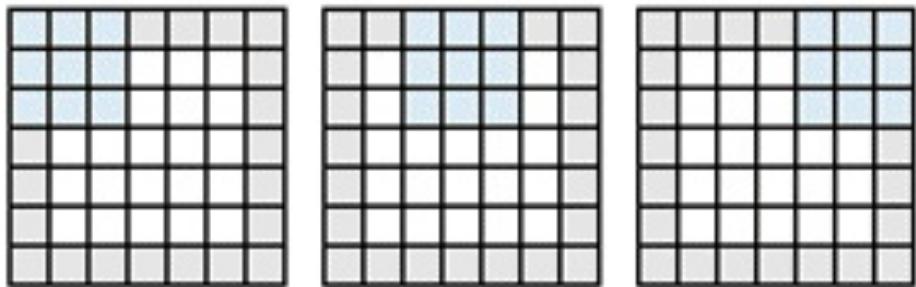


Imagen stride 2



3. Relleno cero: un relleno es una operación de agregar un número correspondiente de filas y columnas a cada lado de los mapas de entidades de entrada. En este caso, la salida tiene la misma dimensión que la entrada.

4. **No linealidad (ReLU)**

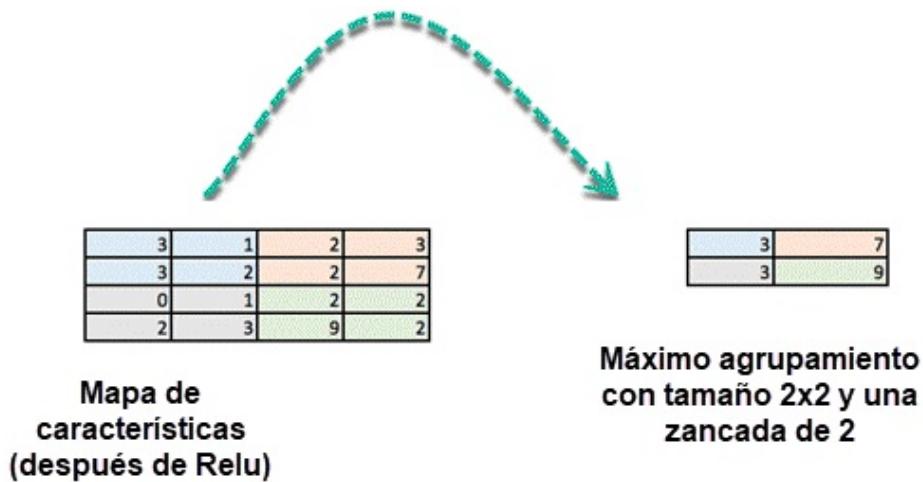
Al final de la operación de convolución, la salida está sujeta a una función de activación para permitir la no linealidad. La función de activación habitual para convnet es el Relu. Todos los píxeles con un valor negativo serán reemplazados por cero.

- **Operación de agrupación de max-pooling**

Este paso es fácil de entender. El propósito de la agrupación es reducir la dimensionalidad de la imagen de entrada. Los pasos se realizan para reducir la complejidad computacional de la operación. Al disminuir la dimensionalidad, la red tiene menos pesos para calcular, por lo que evita el exceso de ajuste.

En esta etapa, debe definir el tamaño y la zancada. Una forma estándar de agrupación de la imagen de entrada es utilizar el valor máximo del mapa de entidades. Mira la imagen de abajo. La “agrupación” mostrará una submatriz de cuatro del mapa de entidades 4×4 y devolverá el valor máximo. La agrupación toma el valor máximo de una matriz 2×2 y luego mueve esta ventana dos píxeles. Por ejemplo, la primera submatriz es

[3,1,3,2], la agrupación devolverá el máximo, que es 3.



Hay otra operación de agrupación como la media.

Esta operación reduce agresivamente el tamaño del mapa de entidades

- Capas completamente conectadas

El último paso consiste en construir una red neuronal artificial tradicional como lo hizo en el tutorial anterior. Conecta todas las neuronas de la capa anterior a la siguiente capa. Se utiliza una función de activación softmax para clasificar el número en la imagen de entrada.

Recapitulación:

La red neuronal convolucional compila diferentes capas antes de hacer una predicción. Una red neuronal tiene:

- Una capa convolucional
- Función de activación de Relu
- Capa de agrupación
- Capa densamente conectada

Las capas convolucionales aplican diferentes filtros en una subregión de

la imagen. La función de activación de Relu añade no linealidad, y las capas de agrupación reducen la dimensionalidad de los mapas de entidades.

Todas estas capas extraen información esencial de las imágenes. Por fin, el mapa de entidades se alimenta a una capa primaria completamente conectada con una función softmax para hacer una predicción.

Tren CNN con TensorFlow

Ahora que usted está familiarizado con el bloque de construcción de un convnets, usted está listo para construir uno con TensorFlow. Utilizaremos el conjunto de datos MNIST para la clasificación de imágenes.

La preparación de datos es la misma que el tutorial anterior. Puede ejecutar los códigos y saltar directamente a la arquitectura de la CNN.

Seguirás los siguientes pasos:

Paso 1: Cargar conjunto de datos

Paso 2: Capa de entrada

Paso 3: Capa convolucional

Paso 4: Capa de agrupación

Paso 5: Segunda capa convolucional y capa de agrupación

Paso 6: Capa densa

Paso 7: Capa de Logit

Paso 1: Cargar conjunto de datos

El conjunto de datos MNIST está disponible con scikit para aprender en esta URL. Descárgalo y guárdelo en Descargas. Puede subirlo con fetch_mldata ('MNIST original').

Crear un conjunto de tren/prueba

Necesita dividir el conjunto de datos con train_test_split

Escalar las entidades

Finalmente, puede escalar la entidad con MinMaxScaler

```
import numpy as np
import tensorflow as tf
from sklearn.datasets import fetch_mldata

#Change USERNAME by the username of your machine
## Windows USER
mnist = fetch_mldata('C:\\\\Users\\\\USERNAME\\\\Downloads\\\\MNIST
original')
## Mac User
mnist = fetch_mldata('/Users/USERNAME/Downloads/MNIST original')

print(mnist.data.shape)
print(mnist.target.shape)
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(mnist.data,
mnist.target, test_size=0.2, random_state=42)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
batch_size =len(X_train)

print(X_train.shape, y_train.shape,y_test.shape )
## resclae
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
# Train
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
# test
X_test_scaled = scaler.fit_transform(X_test.astype(np.float64))
feature_columns = [tf.feature_column.numeric_column('x',
shape=X_train_scaled.shape[1:])]

X_train_scaled.shape[1:]
```

Definir la CNN

Una CNN utiliza filtros en el píxel sin procesar de una imagen para

aprender patrones de detalles comparados con el patrón global con una red neural tradicional. Para construir una CNN, debe definir:

1. Una capa convolucional: aplique n número de filtros al mapa de entidades. Después de la convolución, debe usar una función de activación de Relu para agregar no linealidad a la red.
2. Capa de agrupación: el siguiente paso después de la convolución es reducir el muestreo máximo de la entidad. El propósito es reducir la dimensionalidad del mapa de entidades para evitar el ajuste excesivo y mejorar la velocidad de cálculo. La agrupación máxima es la técnica convencional, que divide los mapas de entidades en subregiones (generalmente con un tamaño de $2x2$) y mantiene solo los valores máximos.
3. Capas completamente conectadas: Todas las neuronas de las capas anteriores están conectadas a las capas siguientes. La CNN clasificará la etiqueta según las entidades de las capas convolucionales y se reducirá con la capa de agrupación.

Arquitectura CNN

- Capa convolucional: Aplica 14 filtros $5x5$ (extrayendo subregiones de $5x5$ píxeles), con función de activación de RelU
- Capa de agrupación: realiza el agrupamiento máximo con un filtro $2x2$ y una zancada de 2 (que especifica que las regiones agrupadas no se superponen)
- Capa convolucional: aplica filtros $36 5x5$, con función de activación de RelU
- Capa de agrupación #2: Una vez más, realiza el agrupamiento máximo con un filtro $2x2$ y zancada de 2
- 1.764 neuronas, con una tasa de regularización de abandono de la escuela de 0.4 (probabilidad de 0.4 de que cualquier elemento dado sea eliminado durante el entrenamiento)
- Capa densa (Capa de Logits): 10 neuronas, una para cada clase

objetivo de dígito (0 – 9).

Hay tres módulos importantes para usar para crear una CNN:

- `conv2d ()`. Construye una capa convolucional bidimensional con el número de filtros, el tamaño del núcleo del filtro, el relleno y la función de activación como argumentos.
- `max_pooling2d ()`. Construye una capa de agrupación bidimensional utilizando el algoritmo de agrupación máxima.
- `dense ()`. Construye una capa densa con las capas y unidades ocultas

Definirá una función para construir la CNN. Veamos en detalle cómo construir cada bloque de construcción antes de envolver todo en la función.

Paso 2: Capa de entrada

```
def cnn_model_fn(features, labels, mode):
    input_layer = tf.reshape(tensor = features["x"], shape =[-1, 28,
28, 1])
```

Debe definir un tensor con la forma de los datos. Para eso, puede usar el módulo `tf.reshape`. En este módulo, debe declarar el tensor para remodelar y la forma del tensor. El primer argumento son las características de los datos, que se definen en el argumento de la función.

Una imagen tiene una altura, un ancho y un canal. El conjunto de datos MNIST es una imagen monocrónica con un tamaño 28x28. Establecimos el tamaño del lote en -1 en el argumento de forma para que tome la forma de las entidades ["x"]. La ventaja es hacer que los hiperparámetros del tamaño del lote se afinen. Si el tamaño del lote se establece en 7, el tensor alimentará 5,488 valores ($28 * 28 * 7$).

Step 3: Convolutional layer

```
# first Convolutional Layer
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=14,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
```

La primera capa convolucional tiene 14 filtros con un tamaño de núcleo de 5x5 con el mismo relleno. El mismo relleno significa que tanto el tensor de salida como el tensor de entrada deben tener la misma altura y anchura. Tensorflow agregará ceros a las filas y columnas para garantizar el mismo tamaño.

Utilice la función de activación de Relu. El tamaño de salida será [28, 28, 14].

Paso 4: Capa de agrupación

El siguiente paso después de la convolución es el cálculo de agrupación. El cálculo de agrupación reducirá la dimensionalidad de los datos. Puede utilizar el módulo max_pooling2d con un tamaño de 2x2 y zancada de 2. Se utiliza la capa anterior como entrada. El tamaño de salida será [batch_size, 14, 14, 14]

```
# first Pooling Layer
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
strides=2)
```

Paso 5: Segunda capa convolucional y capa de agrupación

La segunda capa convolucional tiene 32 filtros, con un tamaño de salida de [batch_size, 14, 14, 32]. La capa de agrupación tiene el mismo tamaño que antes y la forma de salida es [batch_size, 14, 14, 18].

```

conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=36,
    kernel_size=[5, 5],
    padding="same",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
strides=2)

```

Paso 6: Capa densa

Luego, debe definir la capa completamente conectada. El mapa de entidades tiene que ser aplanado antes de conectarse con la capa densa. Puede utilizar el módulo remodelar con un tamaño de $7 * 7 * 36$.

La capa densa conectará 1764 neuronas. Se agrega una función de activación de Relu. Además, agrega un término de regularización de abandono con una tasa de 0,3, lo que significa que el 30 por ciento de los pesos se establecerá en 0. Tenga en cuenta que la deserción tiene lugar solo durante la fase de entrenamiento. La función cnn_model_fn tiene un modo de argumento para declarar si el modelo necesita ser entrenado o para evaluar.

```

pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])

dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.3, training=mode ==
tf.estimator.ModeKeys.TRAIN)

```

Paso 7: Capa de Logit

Finalmente, puede definir la última capa con la predicción del modelo. La forma de salida es igual al tamaño del lote y 10, el número total de imágenes.

```
# Logits Layer  
logits = tf.layers.dense(inputs=dropout, units=10)
```

Puede crear un diccionario que contenga las clases y la probabilidad de cada clase. El módulo `tf.argmax()` devuelve el valor más alto si las capas logit. La función `softmax` devuelve la probabilidad de cada clase.

```
predictions = {  
    # Generate predictions  
    "classes": tf.argmax(input=logits, axis=1),  
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")  
}
```

Sólo desea devolver la predicción diccionaria cuando el modo está establecido en la predicción. Usted agrega estos códigos para despagar las predicciones

```
if mode == tf.estimator.ModeKeys.PREDICT:  
    return tf.estimator.EstimatorSpec(mode=mode,  
predictions=predictions)
```

El siguiente paso consiste en calcular la pérdida del modelo. En el último tutorial, aprendió que la función de pérdida para un modelo multiclase es entropía cruzada. La pérdida se calcula fácilmente con el siguiente código:

```
# Calculate Loss (for both TRAIN and EVAL modes)  
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,  
logits=logits)
```

El paso final es optimizar el modelo, es decir, encontrar los mejores valores de los pesos. Para eso, usa un optimizador de descenso de gradiente con una tasa de aprendizaje de 0.001. El objetivo es minimizar la pérdida

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)  
train_op = optimizer.minimize(  
    loss=loss,
```

```
global_step=tf.train.get_global_step())
```

Ya terminaste con la CNN. Sin embargo, desea mostrar las métricas de rendimiento durante el modo de evaluación. Las métricas de rendimiento para un modelo multiclass son las métricas de precisión. Tensorflow está equipado con una precisión de módulo con dos argumentos, las etiquetas y los valores predichos.

```
eval_metric_ops = {
    "accuracy": tf.metrics.accuracy(labels=labels,
predictions=predictions["classes"])}
return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
eval_metric_ops=eval_metric_ops)
```

Eso es todo. Crea tu primera CNN y estás listo para envolver todo en una función para usarla para entrenar y evaluar el modelo.

```
def cnn_model_fn(features, labels, mode):
    """Model function for CNN."""
    # Input Layer
    input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])

    # Convolutional Layer
    conv1 = tf.layers.conv2d(
        inputs=input_layer,
        filters=32,
        kernel_size=[5, 5],
        padding="same",
        activation=tf.nn.relu)

    # Pooling Layer
    pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],
strides=2)

    # Convolutional Layer #2 and Pooling Layer
    conv2 = tf.layers.conv2d(
        inputs=pool1,
        filters=36,
        kernel_size=[5, 5],
        padding="same",
        activation=tf.nn.relu)
    pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2],
```

```

strides=2)

# Dense Layer
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])
dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.4, training=mode ==
tf.estimator.ModeKeys.TRAIN)

# Logits Layer
logits = tf.layers.dense(inputs=dropout, units=10)

predictions = {
    # Generate predictions (for PREDICT and EVAL mode)
    "classes": tf.argmax(input=logits, axis=1),
    "probabilities": tf.nn.softmax(logits, name="softmax_tensor")
}

if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode=mode,
predictions=predictions)

# Calculate Loss
loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,
logits=logits)

# Configure the Training Op (for TRAIN mode)
if mode == tf.estimator.ModeKeys.TRAIN:
    optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.001)
    train_op = optimizer.minimize(
        loss=loss,
        global_step=tf.train.get_global_step())
    return tf.estimator.EstimatorSpec(mode=mode, loss=loss,
train_op=train_op)

# Add evaluation metrics Evaluation mode
eval_metric_ops = {
    "accuracy": tf.metrics.accuracy(
        labels=labels, predictions=predictions["classes"])}
return tf.estimator.EstimatorSpec(
    mode=mode, loss=loss, eval_metric_ops=eval_metric_ops)

```

Los siguientes pasos son los mismos que los tutoriales anteriores.

En primer lugar, se define un estimador con el modelo CNN.

```
# Create the Estimator
mnist_classifier = tf.estimator.Estimator(
    model_fn=cnn_model_fn, model_dir="train/mnist_convnet_model")
```

Una CNN tarda muchas veces en entrenar, por lo tanto, crea un gancho de registro para almacenar los valores de las capas softmax cada 50 iteraciones.

```
# Set up logging for predictions
tensors_to_log = {"probabilities": "softmax_tensor"}
logging_hook = tf.train.LoggingTensorHook(tensors=tensors_to_log,
                                           every_n_iter=50)
```

Está listo para estimar el modelo. Establecer un tamaño de lote de 100 y barajar los datos. Tenga en cuenta que establecemos pasos de entrenamiento de 16.000, puede tomar mucho tiempo entrenar. Ten paciencia.

```
# Train the model
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train_scaled},
    y=y_train,
    batch_size=100,
    num_epochs=None,
    shuffle=True)
mnist_classifier.train(
    input_fn=train_input_fn,
    steps=16000,
    hooks=[logging_hook])
```

Ahora que el modelo está en tren, puede evaluarlo e imprimir los resultados

```
# Evaluate the model and print results
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_test_scaled},
```

```
y=y_test,  
    num_epochs=1,  
    shuffle=False)  
eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)  
print(eval_results)
```

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-08-05-12:52:41  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from  
train/mnist_convnet_model/model.ckpt-15652  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-08-05-12:52:56  
INFO:tensorflow:Saving dict for global step 15652: accuracy =  
0.9589286, global_step = 15652, loss = 0.13894269  
{'accuracy': 0.9689286, 'loss': 0.13894269, 'global_step': 15652}
```

Con la arquitectura actual, obtiene una precisión del 97%. Puede cambiar la arquitectura, el tamaño del lote y el número de iteraciones para mejorar la precisión. La red neuronal CNN ha tenido un desempeño mucho mejor que ANN o regresión logística. En el tutorial sobre la red neuronal artificial, usted tenía una precisión de 96%, que es menor la CNN. Las actuaciones de la CNN son impresionantes con una imagen más grande **conjunto**, tanto en términos de cálculo de velocidad como de precisión.

Resumen

Una red neuronal convolucional funciona muy bien para evaluar la imagen. Este tipo de arquitectura es dominante para reconocer objetos de una imagen o vídeo.

Para crear una CNN, debe seguir seis pasos:

Paso 1: Capa de entrada:

Este paso cambia la forma de los datos. La forma es igual a la raíz

cuadrada del número de píxeles. Por ejemplo, si una imagen tiene 156 píxeles, entonces la forma es 26x26. Debe especificar si la imagen tiene color o no. Si es así, entonces tenía 3 a la forma- 3 para RGB-, de lo contrario 1.

```
input_layer = tf.reshape(tensor = features["x"], shape =[-1, 28, 28, 1])
```

Paso 2: Capa convolucional

A continuación, debe crear las capas convolucionales. Aplicar diferentes filtros para permitir que la red aprenda característica importante. Especifique el tamaño del núcleo y la cantidad de filtros.

```
conv1 = tf.layers.conv2d(  
    inputs=input_layer,  
    filters=14,  
    kernel_size=[5, 5],  
    padding="same",  
    activation=tf.nn.relu)
```

Paso 3: capa de agrupación

En el tercer paso, se agrega una capa de agrupación. Esta capa disminuye el tamaño de la entrada. Lo hace tomando el valor máximo de la submatriz a. Por ejemplo, si la submatriz es [3,1,3,2], la agrupación devolverá el máximo, que es 3.

```
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2],  
strides=2)
```

Paso 4: Agregar capa convolucional y capa de agrupación

En este paso, puede agregar tanto como desee capas conv y capas de agrupación. Google utiliza arquitectura con más de 20 capas conv.

Paso 5: Capa densa

El paso 5 aplanar el anterior para crear unas capas completamente conectadas. En este paso, puede utilizar diferentes funciones de activación y agregar un efecto de abandono.

```
pool2_flat = tf.reshape(pool2, [-1, 7 * 7 * 36])

dense = tf.layers.dense(inputs=pool2_flat, units=7 * 7 * 36,
activation=tf.nn.relu)
dropout = tf.layers.dropout(
    inputs=dense, rate=0.3, training=mode ==
tf.estimator.ModeKeys.TRAIN)
```

Paso 6: Capa de Logit

El último paso es la predicción.

```
logits = tf.layers.dense(inputs=dropout, units=10)
```

Capítulo 19: Autoencoder con TensorFlow

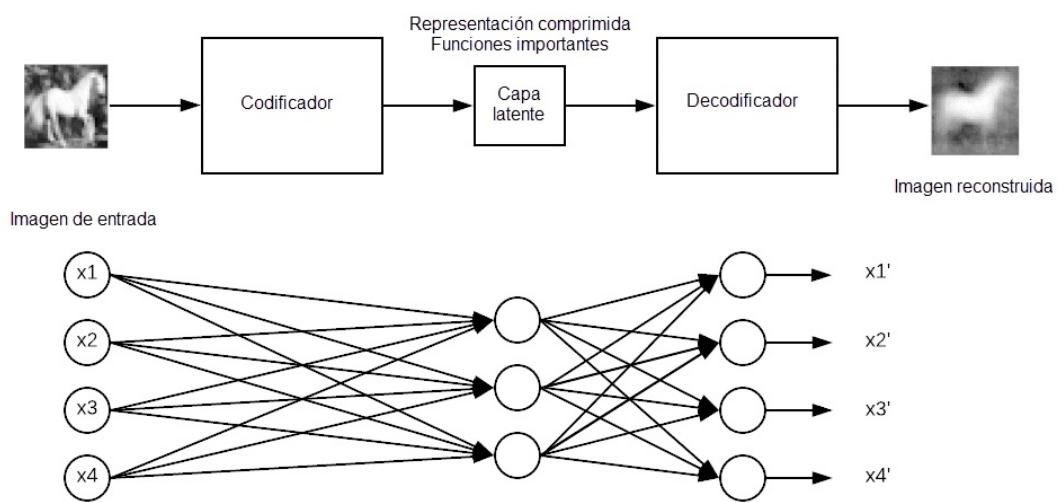
¿Qué es un autocodificador?

Un autocodificador es una gran herramienta para recrear una entrada. En una palabra simple, la máquina toma, digamos una imagen, y puede producir una imagen estrechamente relacionada. La entrada en este tipo de red neuronal está desetiquetada, lo que significa que la red es capaz de aprender sin supervisión. Más precisamente, la red codifica la entrada para centrarse solo en la entidad más crítica. Esta es una de las razones por las que el autoencoder es popular para la reducción de dimensionalidad. Además, los autocodificadores se pueden utilizar para producir **modelos de aprendizaje generativo**. Por ejemplo, la red neuronal puede ser entrenada con un conjunto de caras y luego puede producir nuevas caras.

¿Cómo funciona Autoencoder?

El propósito de un autocodificador es producir una aproximación de la entrada centrándose únicamente en las características esenciales. Puede pensar por qué no simplemente aprender a copiar y pegar la entrada para producir la salida. De hecho, un autocodificador es un conjunto de restricciones que obligan a la red a aprender nuevas formas de representar los datos, diferentes de simplemente copiar la salida.

Un autocodificador típico se define con una entrada, una representación interna y una salida (una aproximación de la entrada). El aprendizaje ocurre en las capas adjuntas a la representación interna. De hecho, hay dos bloques principales de capas que se parecen a una red neuronal tradicional. La ligera diferencia es que la capa que contiene la salida debe ser igual a la entrada. En la imagen de abajo, la entrada original entra en el primer bloque llamado **codificador**. Esta representación interna comprime (reduce) el tamaño de la entrada. En el segundo bloque se produce la reconstrucción de la entrada. Esta es la fase de decodificación.



El modelo actualizará los pesos minimizando la función de pérdida. El modelo se penaliza si la salida de reconstrucción es diferente de la entrada.

Concretamente, imagina una imagen con un tamaño de 50x50 (es decir, 250 píxeles) y una red neuronal con una sola capa oculta compuesta por cien neuronas. El aprendizaje se realiza en un mapa de entidades que es dos veces más pequeño que la entrada. Significa que la red necesita encontrar una manera de reconstruir 250 píxeles con sólo un vector de neuronas igual a 100.

Ejemplo de Autocodificador apilado

En este tutorial, aprenderá a utilizar un autocodificador apilado. La arquitectura es similar a una red neuronal tradicional. La entrada va a una capa oculta para ser comprimida, o reducir su tamaño, y luego llega a las capas de reconstrucción. El objetivo es producir una imagen de salida tan cercana como la original. El modelo tiene que aprender una manera de lograr su tarea bajo un conjunto de restricciones, es decir, con una dimensión más baja.

Hoy en día, los autocodificadores se utilizan principalmente para retocar una imagen. Imagine una imagen con araños; un humano todavía es capaz de reconocer el contenido. La idea de retocar el autocodificador es agregar ruido a la imagen para forzar a la red a aprender el patrón detrás de los datos.

La otra familia útil de autocodificador es el autocodificador variacional. Este tipo de red puede generar nuevas imágenes. Imagina que entrenas una red con la imagen de un hombre; tal red puede producir nuevas caras.

Crear un autocodificador con TensorFlow

En este tutorial, aprenderá a construir un autocodificador apilado para reconstruir una imagen.

Utilizará el conjunto de datos CIFAR-10 que contiene 60000 imágenes en color 32x32. El conjunto de datos ya está dividido entre 50000 imágenes para entrenamiento y 10000 para pruebas. Hay hasta diez clases:

- Avión
- Automóvil
- Pájaro
- Gato
- Ciervo
- Perro
- Rana
- Caballo
- Barco
- Camión

Es necesario descargar las imágenes en esta URL
<https://www.cs.toronto.edu/~kriz/cifar.html> y descomprimirlas. La carpeta for-10-batches-py contiene cinco lotes de datos con 10000 imágenes cada uno en un orden aleatorio.

Antes de crear y entrenar su modelo, debe aplicar un poco de procesamiento de datos. Procederá de la siguiente manera:

1. Importar los datos
2. Convertir los datos a formato blanco/negra

3. Anexar todos los lotes
4. Construir el conjunto de datos de formación
5. Construir un visualizador de imágenes

Preprocesamiento de imágenes

Paso 1) Importe los datos.

De acuerdo con el sitio web oficial, puede cargar los datos con el siguiente código. El código cargará los datos en un diccionario con el **dato** y el **etiqueta**. Tenga en cuenta que el código es una función.

```
import numpy as np
import tensorflow as tf
import pickle
def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding='latin1')
    return dict
```

Paso 2) Convertir los datos a formato blanco/negra

Para simplificar, convertirá los datos en una escala de grises. Es decir, con una sola dimensión contra tres para la imagen de colores. La mayor parte de la red neuronal funciona solo con una entrada de dimensión.

```
def grayscale(im):
    return im.reshape(im.shape[0], 3, 32,
32).mean(1).reshape(im.shape[0], -1)
```

Paso 3) Anexar todos los lotes

Ahora que se crean ambas funciones y se carga el dataset, puede escribir un bucle para anexar los datos en la memoria. Si comprueba cuidadosamente, el archivo de descompresión con los datos se denomina data_batch_ con un número del 1 al 5. Puede recorrer los archivos y

anexarlos a los datos.

Cuando termine este paso, convertirá los datos de colores a un formato de escala de grises. Como puede ver, la forma de los datos es 50000 y 1024. Los 32 * 32 píxeles ahora se aplanan a 2014.

```
# Load the data into memory
data, labels = [], []
## Loop over the b
for i in range(1, 6):
    filename = './cifar-10-batches-py/data_batch_' + str(i)
    open_data = unpickle(filename)
    if len(data) > 0:
        data = np.vstack((data, open_data['data']))
        labels = np.hstack((labels, open_data['labels']))
    else:
        data = open_data['data']
        labels = open_data['labels']

data = grayscale(data)
x = np.matrix(data)
y = np.array(labels)
print(x.shape)
(50000, 1024)
```

Nota: Cambio './cifar-10-batches-py/data_batch_' a la ubicación real del archivo. Por ejemplo, para la máquina Windows, la ruta podría ser nombre de archivo = 'E:\cifar-10-batches-py\data_batch_' + str(i)

Paso 4) Construir el conjunto de datos de formación

Para hacer el entrenamiento más rápido y fácil, entrenará a un modelo solo en las imágenes del caballo. Los caballos son la séptima clase en los datos de la etiqueta. Como se menciona en la documentación del conjunto de datos CIFAR-10, cada clase contiene 5000 imágenes. Puede imprimir la forma de los datos para confirmar que hay 5.000 imágenes con 1024 columnas.

```
horse_i = np.where(y == 7)[0]
```

```
horse_x = x[horse_i]
print(np.shape(horse_x))
(5000, 1024)
```

Paso 5) Construir un visualizador de imágenes

Finalmente, se construye una función para trazar las imágenes.

Necesitará esta función para imprimir la imagen reconstruida desde el autocodificador.

Una forma fácil de imprimir imágenes es usar el objeto imshow de la biblioteca matplotlib. Tenga en cuenta que, debe convertir la forma de los datos de 1024 a 32 * 32 (es decir, el formato de una imagen).

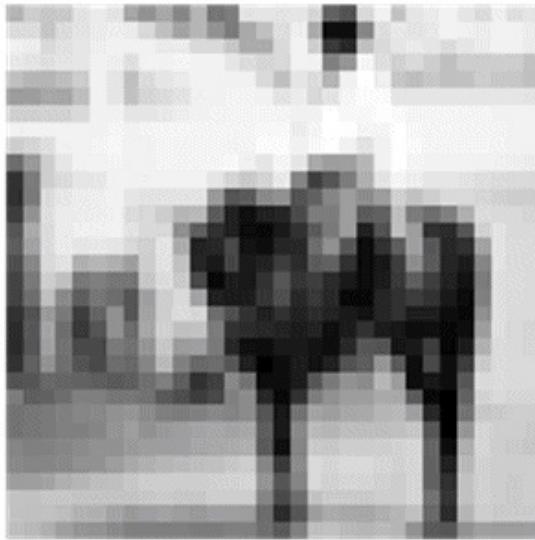
```
# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
def plot_image(image, shape=[32, 32], cmap = "Greys_r"):
    plt.imshow(image.reshape(shape),
cmap=cmap, interpolation="nearest")
    plt.axis("off")
```

La función toma 3 argumentos:

- Imagen: la entrada
- Forma: lista, la dimensión de la imagen
- Cmap: elige el mapa de colores. Por defecto, gris

Puede intentar trazar la primera imagen en el conjunto de datos. Debería ver a un hombre en un caballo.

```
plot_image(horse_x[1], shape=[32, 32], cmap = "Greys_r")
```



Establecer estimador de conjunto de datos

Muy bien, ahora que el conjunto de datos está listo para usar, puede comenzar a usar Tensorflow. Antes de construir el modelo, usemos el estimador de conjunto de datos de Tensorflow para alimentar la red.

Construirá un conjunto de datos con el estimador TensorFlow. Para refrescar su mente, necesita usar:

- from_tensor_slices
- Repetir
- lote

El código completo para construir el conjunto de datos es:

```
dataset =  
tf.data.Dataset.from_tensor_slices(x).repeat().batch(batch_size)
```

Tenga en cuenta que, x es un marcador de posición con la siguiente forma:

- [None, n_inputs]: Establezca en Ninguno porque el número de fuente de imágenes a la red es igual al tamaño del lote.

para obtener más detalles, consulte el tutorial sobre regresión lineal.

Después de eso, debe crear el iterador. Sin esta línea de código, ningún dato pasará por la tubería.

```
iter = dataset.make_initializable_iterator() # create the
iteratorfeatures = iter.get_next()
```

Ahora que la tubería está lista, puede verificar si la primera imagen es la misma que antes (es decir, un hombre en un caballo).

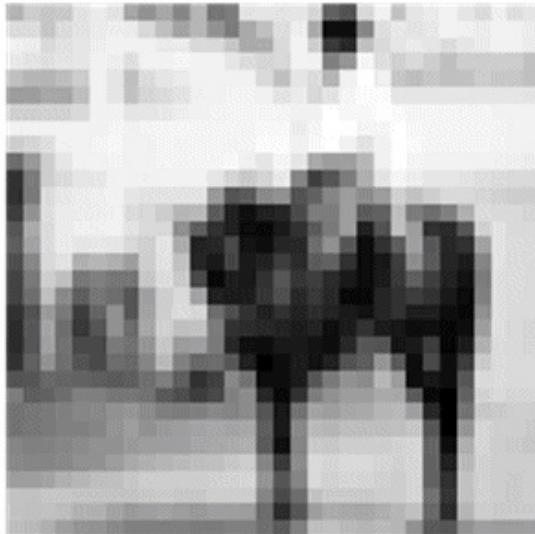
Establezca el tamaño del lote en 1 porque solo desea alimentar el dataset con una imagen. Puede ver la dimensión de los datos con print (sess.run (features) .shape). Es igual a (1, 1024). 1 significa que sólo una imagen con 1024 se alimenta cada una. Si el tamaño del lote se establece en dos, entonces dos imágenes pasarán por la canalización. (No cambie el tamaño del lote. De lo contrario, arrojará un error. Sólo una imagen a la vez puede ir a la función plot_image () .

```
## Parameters
n_inputs = 32 * 32
BATCH_SIZE = 1
batch_size = tf.placeholder(tf.int64)

# using a placeholder
x = tf.placeholder(tf.float32, shape=[None, n_inputs])
## Dataset
dataset =
tf.data.Dataset.from_tensor_slices(x).repeat().batch(batch_size)
iter = dataset.make_initializable_iterator() # create the iterator
features = iter.get_next()

## Print the image
with tf.Session() as sess:
    # feed the placeholder with data
    sess.run(iter.initializer, feed_dict={x: horse_x,
```

```
batch_size: BATCH_SIZE})  
print(sess.run(features).shape)  
plot_image(sess.run(features), shape=[32, 32], cmap =  
"Greys_r")  
(1, 1024)
```

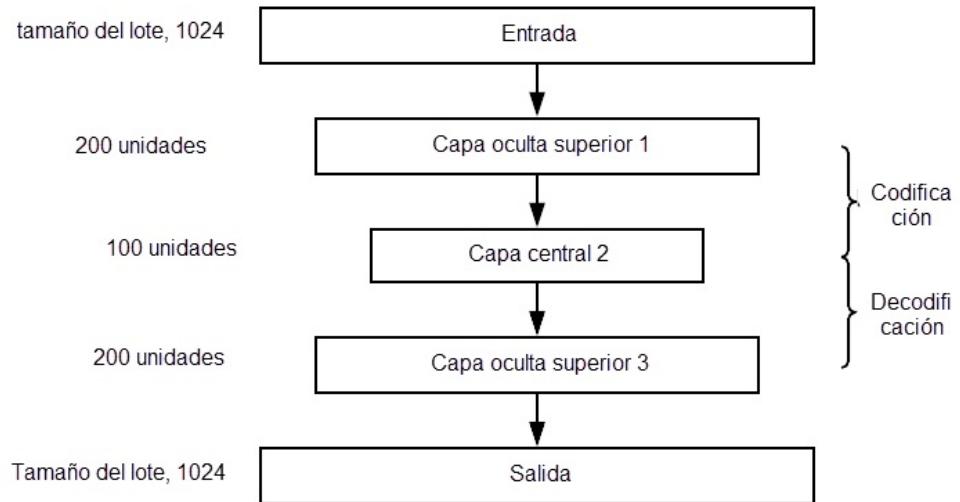


Construir la red

Es hora de construir la red. Entrenará un autocodificador apilado, es decir, una red con múltiples capas ocultas.

Su red tendrá una capa de entrada con 1024 puntos, es decir, 32x32, la forma de la imagen.

El bloque codificador tendrá una capa oculta superior con 300 neuronas, una capa central con 150 neuronas. El bloque decodificador es simétrico con el codificador. Puede visualizar la red en la imagen de abajo. Tenga en cuenta que puede cambiar los valores de las capas ocultas y centrales.



La construcción de un autocodificador es muy similar a cualquier otro modelo de aprendizaje profundo.

Construirá el modelo siguiendo estos pasos:

1. Definir los parámetros
2. Definir las capas
3. Definir la arquitectura
4. Definir la optimización
5. Ejecutar el modelo
6. Evaluar el modelo

En la sección anterior, aprendió cómo crear una canalización para alimentar el modelo, por lo que no es necesario crear una vez más el dataset. Construirá un autocodificador con cuatro capas. Usan la inicialización de Xavier. Esta es una técnica para establecer los pesos iniciales iguales a la varianza tanto de la entrada como de la salida. Finalmente, usa la función de activación elu. Usted regulariza la función de pérdida con el regularizador L2.

Paso 1) Definir los parámetros

El primer paso implica definir el número de neuronas en cada capa, la tasa de aprendizaje y el hiperparámetro del regularizador.

Antes de eso, importa la función parcialmente. Es un mejor método para definir los parámetros de las capas densas. El siguiente código define los valores de la arquitectura de autoencoder. Como se ha indicado anteriormente, el autocodificador tiene dos capas, con 300 neuronas en las primeras capas y 150 en las segundas capas. Sus valores se almacenan en n_hidden_1 y n_hidden_2.

Debe definir la tasa de aprendizaje y el hiperparámetro L2. Los valores se almacenan en learning_rate y l2_reg

```
from functools import partial

## Encoder
n_hidden_1 = 300
n_hidden_2 = 150 # codings

## Decoder
n_hidden_3 = n_hidden_1
n_outputs = n_inputs

learning_rate = 0.01
l2_reg = 0.0001
```

La técnica de inicialización de Xavier se llama con el objeto xavier_initializer del estimador contrib. En el mismo estimador, puede agregar el regularizador con l2_regularizer

```
## Define the Xavier initialization
xav_init = tf.contrib.layers.xavier_initializer()
## Define the L2 regularizer
l2_regularizer = tf.contrib.layers.l2_regularizer(l2_reg)
```

Paso 2) Definir las capas

Se han establecido todos los parámetros de las capas densas; puede

empaquetar todo en la variable dense_layer utilizando el objeto parcialmente. dense_layer que utiliza la activación de ELU, la inicialización de Xavier y la regularización L2.

```
## Create the dense layer
dense_layer = partial(tf.layers.dense,
                      activation=tf.nn.elu,
                      kernel_initializer=xav_init,
                      kernel_regularizer=l2_regularizer)
```

Paso 3) Definir la arquitectura

Si observa la imagen de la arquitectura, observa que la red acumula tres capas con una capa de salida. En el siguiente código, conecta las capas apropiadas. Por ejemplo, la primera capa calcula el producto de punto entre las entidades de matriz de entradas y las matrices que contienen los pesos 300. Después de calcular el producto de punto, la salida pasa a la función de activación de Elu. La salida se convierte en la entrada de la siguiente capa, por eso la usa para calcular hidden_2 y así sucesivamente. La multiplicación de matrices es la misma para cada capa porque se utiliza la misma función de activación. Tenga en cuenta que la última capa, salidas, no aplica una función de activación. Tiene sentido porque esta es la entrada reconstruida

```
## Make the mat mul
hidden_1 = dense_layer(features, n_hidden_1)
hidden_2 = dense_layer(hidden_1, n_hidden_2)
hidden_3 = dense_layer(hidden_2, n_hidden_3)
outputs = dense_layer(hidden_3, n_outputs, activation=None)
```

Paso 4) Definir la optimización

El último paso es construir el optimizador. Se utiliza el error cuadrado medio como una función de pérdida. Si recuerda el tutorial sobre regresión lineal, sabe que el MSE se calcula con la diferencia entre la salida prevista y la etiqueta real. Aquí, la etiqueta es la entidad porque el

modelo intenta reconstruir la entrada. Por lo tanto, desea la media de la suma de la diferencia del cuadrado entre la salida prevista y la entrada. Con TensorFlow, puede codificar la función de pérdida de la siguiente manera:

```
loss = tf.reduce_mean(tf.square(outputs - features))
```

Luego, debe optimizar la función de pérdida. Utilice el optimizador de Adam para calcular los gradientes. La función objetivo es minimizar la pérdida.

```
## Optimize
loss = tf.reduce_mean(tf.square(outputs - features))
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

Una configuración más antes de entrenar el modelo. Desea utilizar un tamaño de lote de 150, es decir, alimentar la tubería con 150 imágenes cada iteración. Debe calcular el número de iteraciones manualmente. Esto es trivial de hacer:

Si desea pasar 150 imágenes cada vez y sabe que hay 5000 imágenes en el conjunto de datos, el número de iteraciones es igual a. En python puede ejecutar los siguientes códigos y asegurarse de que la salida sea 33:

```
BATCH_SIZE = 150
### Number of batches : length dataset / batch size
n_batches = horse_x.shape[0] // BATCH_SIZE
print(n_batches)
33
```

Paso 5) Ejecute el modelo

Por último, pero no menos importante, entrena el modelo. Estás entrenando el modelo con 100 épocas. Es decir, el modelo verá 100 veces las imágenes para pesos optimizados.

Ya está familiarizado con los códigos para entrenar un modelo en Tensorflow. La ligera diferencia es canalizar los datos antes de ejecutar el entrenamiento. De esta manera, el modelo entrena más rápido.

Usted está interesado en imprimir la pérdida después de diez épocas para ver si el modelo está aprendiendo algo (es decir, la pérdida está disminuyendo). El entrenamiento dura de 2 a 5 minutos, dependiendo del hardware de la máquina.

```
## Set params
n_epochs = 100

## Call Saver to save the model and re-use it later during
evaluation
saver = tf.train.Saver()

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    # initialise iterator with train data
    sess.run(iterator.initializer, feed_dict={x: horse_x,
                                                batch_size: BATCH_SIZE})
    print('Training...')
    print(sess.run(features).shape)
    for epoch in range(n_epochs):
        for iteration in range(n_batches):
            sess.run(train)
        if epoch % 10 == 0:
            loss_train = loss.eval() # not shown
            print("\r{}.{format(epoch), "Train MSE:", loss_train)
            saver.save(sess, "./my_model_all_layers.ckpt")
        save_path = saver.save(sess, "./model.ckpt")
        print("Model saved in path: %s" % save_path)
Training...
(150, 1024)
0 Train MSE: 2934.455
10 Train MSE: 1672.676
20 Train MSE: 1514.709
30 Train MSE: 1404.3118
40 Train MSE: 1425.058
50 Train MSE: 1479.0631
60 Train MSE: 1609.5259
70 Train MSE: 1482.3223
```

```
80 Train MSE: 1445.7035
90 Train MSE: 1453.8597
Model saved in path: ./model.ckpt
```

Paso 6) Evaluar el modelo

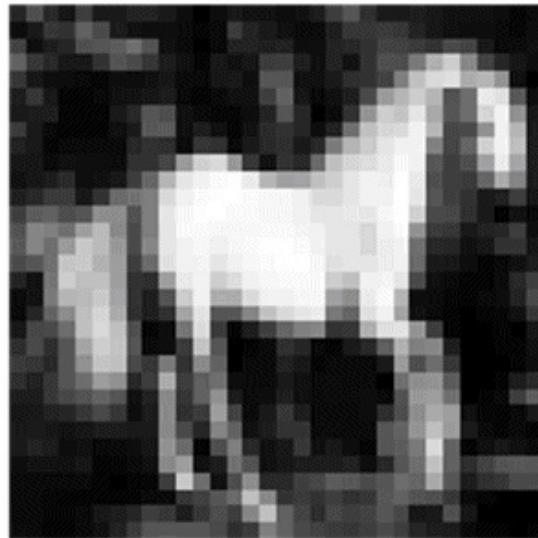
Ahora que tienes tu modelo entrenado, es hora de evaluarlo. Debe importar el sert de prueba del archivo /cifar-10-batches-py/.

```
test_data = unpickle('./cifar-10-batches-py/test_batch')
test_x = grayscale(test_data['data'])
#test_labels = np.array(test_data['labels'])
```

NOTA: Para una máquina Windows, el código se convierte en test_data = unpickle(r"E:\cifar-10-batches-py\test_batch")

Usted puede tratar de imprimir las imágenes 13, que es un caballo

```
plot_image(test_x[13], shape=[32, 32], cmap = "Greys_r")
```



Para evaluar el modelo, utilizará el valor de píxel de esta imagen y verá si el codificador puede reconstruir la misma imagen después de reducir 1024 píxeles. Tenga en cuenta que define una función para evaluar el

modelo en diferentes imágenes. El modelo debe funcionar mejor solo en caballos.

La función toma dos argumentos:

- df: Importar los datos de prueba
- image_number: indicar qué imagen importar

La función se divide en tres partes:

1. Cambiar la forma de la imagen a la dimensión correcta, es decir, 1, 1024
2. Alimentar el modelo con la imagen invisible, codificar/descodificar la imagen
3. Imprimir la imagen real y reconstruida

```
def reconstruct_image(df, image_number = 1):
    ## Part 1: Reshape the image to the correct dimension i.e 1,
    1024
    x_test = df[image_number]
    x_test_1 = x_test.reshape((1, 32*32))

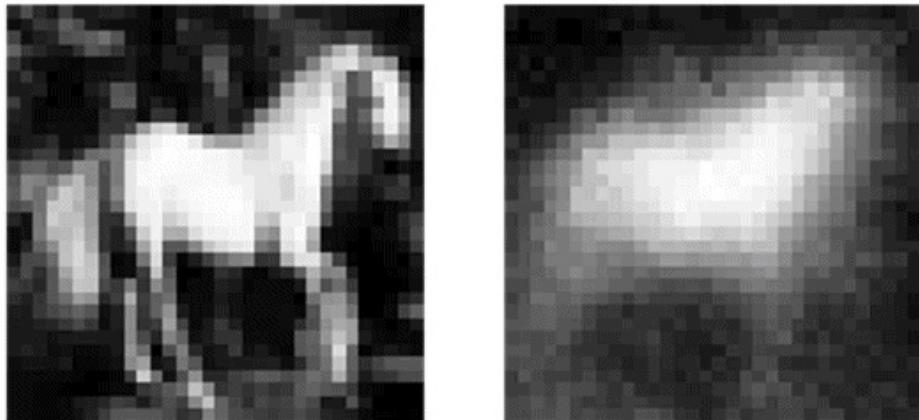
    ## Part 2: Feed the model with the unseen image, encode/decode
    the image
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(iterator.initializer, feed_dict={x: x_test_1,
                                                   batch_size: 1})
    ## Part 3: Print the real and reconstructed image
    # Restore variables from disk.
    saver.restore(sess, "./model.ckpt")
    print("Model restored.")
    # Reconstruct image
    outputs_val = outputs.eval()
    print(outputs_val.shape)
    fig = plt.figure()
    # Plot real
    ax1 = fig.add_subplot(121)
    plot_image(x_test_1, shape=[32, 32], cmap = "Greys_r")
    # Plot estimated
```

```
ax2 = fig.add_subplot(122)
plot_image(outputs_val, shape=[32, 32], cmap = "Greys_r")
plt.tight_layout()
fig = plt.gcf()
```

Ahora que la función de evaluación está definida, puede echar un vistazo a la imagen reconstruida número trece

```
reconstruct_image(df =test_x, image_number = 13)
```

```
INFO:tensorflow:Restoring parameters from ./model.ckpt
Model restored.
(1, 1024)
```



Resumen

El propósito principal de un autocodificador es comprimir los datos de entrada y luego descomprimirlos en una salida que se parece mucho a los datos originales.

La arquitectura de un autocodificador simétrico con una capa pivotante denominada capa central.

Puede crear el autocodificador usando:

- Parcial: para crear las capas densas con la configuración típica:

- ```
tf.layers.dense,
 activation=tf.nn.elu,
 kernel_initializer=xav_init,
 kernel_regularizer=l2_regularizer
```

- `dense_layer ()`: para hacer la multiplicación de la matriz

puede definir la función de pérdida y la optimización con:

```
loss = tf.reduce_mean(tf.square(outputs - features))
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(loss)
```

Última ejecución de una sesión para entrenar el modelo.

# **Capítulo 20: RNN (red neuronal recurrente)**

## **TensorFlow**

### **¿Qué necesitamos un RNN?**

La estructura de una Red Neural Artificial es relativamente simple y se refiere principalmente a la multiplicación de la maduración. Durante el primer paso, las entradas se multiplican por pesos inicialmente aleatorios, y el sesgo, transformado con una función de activación y los valores de salida se utilizan para hacer una predicción. Este paso da una idea de cuán lejos está la red de la realidad.

La métrica aplicada es la pérdida. Cuanto mayor sea la función de pérdida, más tonto es el modelo. Para mejorar el conocimiento de la red, se requiere cierta optimización ajustando los pesos de la red. El descenso del gradiente estocástico es el método empleado para cambiar los valores de los pesos en la dirección de los derechos. Una vez realizado el ajuste, la red puede utilizar otro lote de datos para probar sus nuevos conocimientos.

El error, afortunadamente, es menor que antes, pero no lo suficientemente pequeño. El paso de optimización se realiza iterativamente hasta que se minimiza el error, es decir, no se puede extraer más información.

El problema con este tipo de modelo es que no tiene memoria. Significa que la entrada y la salida son independientes. En otras palabras, el modelo no se preocupa por lo que vino antes. Se plantea alguna pregunta

cuando se necesita predecir series temporales o oraciones porque la red necesita tener información sobre los datos históricos o las palabras pasadas.

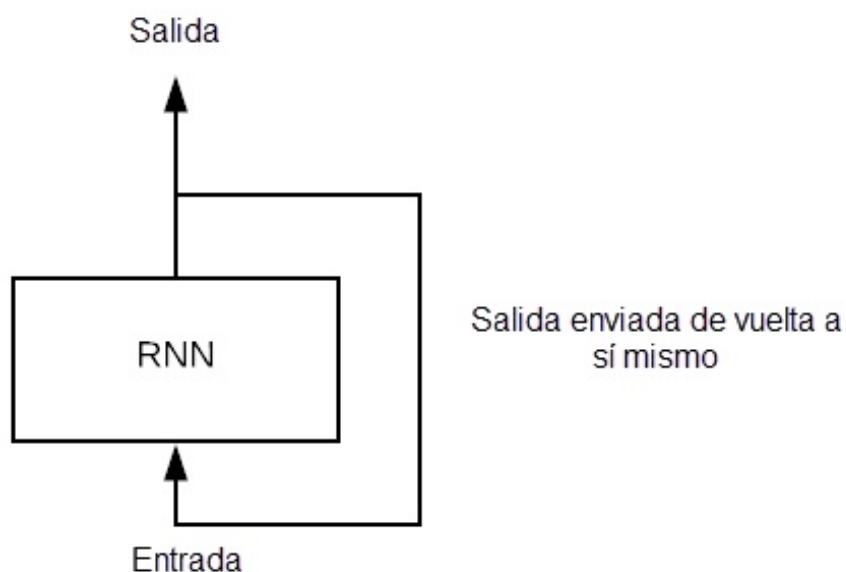
Para superar este problema, se ha desarrollado un nuevo tipo de arquitectura: Red neuronal recurrente (RNN en adelante)

# ¿Qué es RNN?

Una red neuronal recurrente parece bastante similar a una red neuronal tradicional excepto que se agrega un estado de memoria a las neuronas. El cálculo para incluir una memoria es simple.

Imagine un modelo simple con una sola neurona alimentada por un lote de datos. En una red neuronal tradicional, el modelo produce la salida multiplicando la entrada con el peso y la función de activación. Con un RNN, esta salida se envía de vuelta a sí mismo número de tiempo. Llamamos **timestep** la cantidad de tiempo que la salida se convierte en la entrada de la siguiente multiplicación de matriz.

Por ejemplo, en la imagen de abajo, puede ver que la red está compuesta por una neurona. La red calcula la multiplicación de matrices entre la entrada y el peso y añade no linealidad con la función de activación. Se convierte en la salida en t-1. Esta salida es la entrada de la segunda multiplicación de matrices.

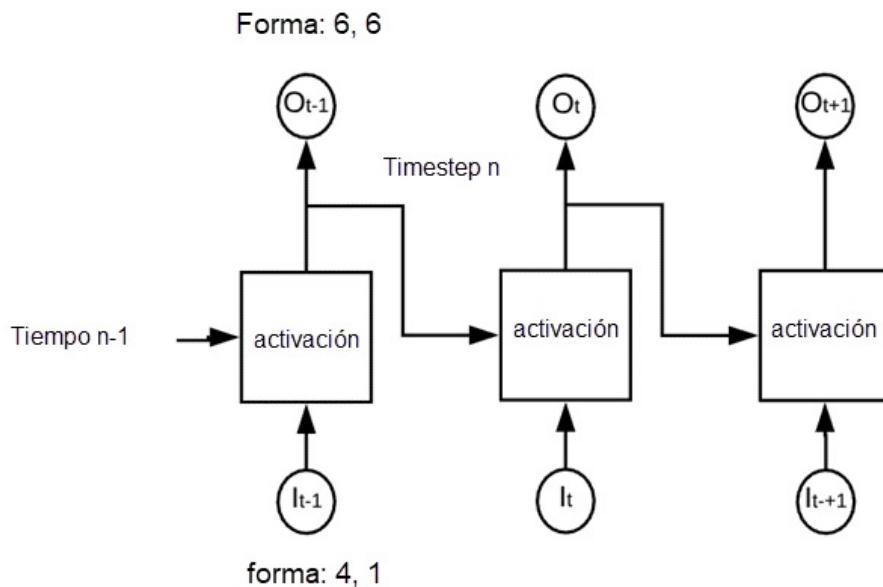


A continuación, codificamos un simple RNN en tensorflow para entender el paso y también la forma de la salida.

La red se compone de:

- Cuatro entradas
- Seis neuronas
- Pasos de dos tiempos

La red procederá como se muestra en la imagen de abajo.



La red se llama 'recurrente' porque realiza la misma operación en cada casilla de activación. La red calculó los pesos de las entradas y la salida anterior antes de utilizar una función de activación.

```
import numpy as np
import tensorflow as tf
n_inputs = 4
n_neurons = 6
n_timesteps = 2
The data is a sequence of a number from 0 to 9 and divided into
three batches of data.
Data
```

```
x_batch = np.array([
 [[0, 1, 2, 5], [9, 8, 7, 4]], # Batch 1
 [[3, 4, 5, 2], [0, 0, 0, 0]], # Batch 2
 [[6, 7, 8, 5], [6, 5, 4, 2]], # Batch 3
])
```

Podemos construir la red con un marcador de posición para los datos, la etapa recurrente y la salida.

## 1. Definir el marcador de posición para los datos

```
x = tf.placeholder(tf.float32, [None, n_timesteps, n_inputs])
```

Aquí:

- Ninguno: Desconocido y tomará el tamaño del lote
- n\_timesteps: Número de tiempo que la red enviará la salida de vuelta a la neurona
- n\_inputs: Número de entradas por lote

## 2. Definir la red recurrente

Como se mencionó en la imagen de arriba, la red se compone de 6 neuronas. La red calculará dos productos de punto:

- Datos de entrada con el primer conjunto de pesos (es decir, 6: igual al número de neuronas)
- Salida anterior con un segundo conjunto de pesos (es decir, 6: correspondiente al número de salida)

Tenga en cuenta que, durante el primer feedforward, los valores de la salida anterior son iguales a ceros porque no tenemos ningún valor disponible.

El objeto para construir un RNN es `tf.contrib.rnn.BasicRNNCell` con el argumento `num_units` para definir el número de entrada

```
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
```

Ahora que la red está definida, puede calcular las salidas y los estados

```
outputs, states = tf.nn.dynamic_rnn(basic_cell, X,
dtype=tf.float32)
```

Este objeto utiliza un bucle interno para multiplicar las matrices el número apropiado de veces.

Tenga en cuenta que la neurona recurrente es una función de todas las entradas de los pasos de tiempo anteriores. Así es como la red construye su propia memoria. La información de la hora anterior puede propagarse en el futuro. Esta es la magia de la red neuronal recurrente

```
Define the shape of the tensor
X = tf.placeholder(tf.float32, [None, n_timesteps, n_inputs])
Define the network
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
outputs, states = tf.nn.dynamic_rnn(basic_cell, X,
dtype=tf.float32)
init = tf.global_variables_initializer()
init = tf.global_variables_initializer()
with tf.Session() as sess:
 init.run()
 outputs_val = outputs.eval(feed_dict={X: X_batch})
print(states.eval(feed_dict={X: X_batch}))
[[0.38941205 -0.9980438 0.99750966 0.7892596 0.9978241
0.9999997]
 [0.61096436 0.7255889 0.82977575 -0.88226104 0.29261455
-0.15597084]
 [0.62091285 -0.87023467 0.99729395 -0.58261937 0.9811445
0.99969864]]
```

A efectos explicativos, se imprimen los valores del estado anterior. La salida impresa arriba muestra la salida del último estado. Ahora imprime toda la salida, puede notar que los estados son la salida anterior de cada lote. Es decir, la salida anterior contiene la información sobre toda la secuencia.e

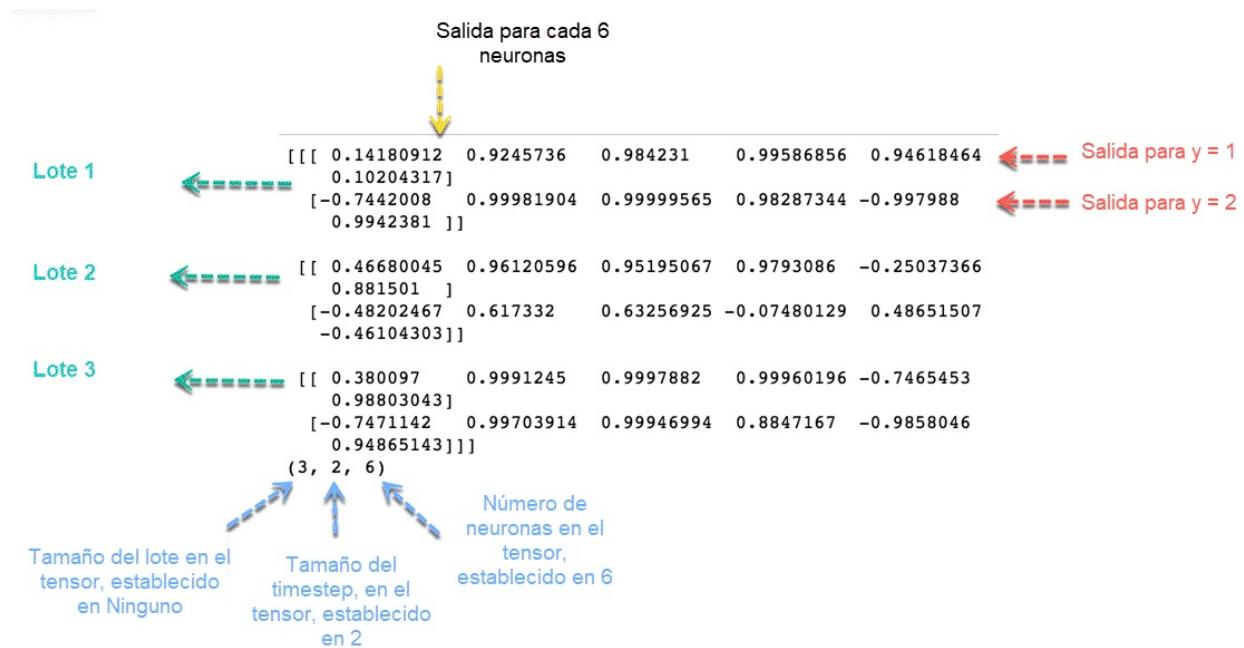
```

print(outputs_val)
print(outputs_val.shape)
[[[-0.75934666 -0.99537754 0.9735819 -0.9722234 -0.14234993
 -0.9984044]
 [0.99975264 -0.9983206 0.9999993 -1. -0.9997506
 -1.]]

[[0.97486496 -0.98773265 0.9969686 -0.99950117 -0.7092863
 -0.99998885]
 [0.9326837 0.2673438 0.2808514 -0.7535883 -0.43337247
 0.5700631]]

[[0.99628735 -0.9998728 0.99999213 -0.99999976 -0.9884324
 -1.]
 [0.99962527 -0.9467421 0.9997403 -0.99999714 -0.99929446
 -0.9999795]]]
(3, 2, 6)

```



La salida tiene la forma de (3, 2, 6):

- 3: Número de lotes
- 2: Número del timestep
- 6: Número de neuronas

La optimización de una red neuronal recurrente es idéntica a una red neuronal tradicional. Verá con más detalle cómo codificar la optimización en la siguiente parte de este tutorial.

## Aplicaciones de RNN

RNN tiene múltiples usos, especialmente cuando se trata de predecir el futuro. En la industria financiera, RNN puede ser útil para predecir los precios de las acciones o el signo de la dirección del mercado de valores (es decir, positivo o negativo).

RNN es útil para un automóvil autónomo ya que puede evitar un accidente automovilístico anticipando la trayectoria del vehículo.

RNN es ampliamente utilizado en análisis de texto, subtítulos de imágenes, análisis de sentimientos y traducción automática. Por ejemplo, uno puede utilizar una revisión de película para entender la sensación que el espectador percibió después de ver la película. Automatizar esta tarea es muy útil cuando la compañía de cine no tiene tiempo suficiente para revisar, etiquetar, consolidar y analizar las revisiones. La máquina puede hacer el trabajo con un mayor nivel de precisión.

## Limitaciones de RNN

En teoría, RNN se supone que lleva la información al tiempo. Sin embargo, es bastante difícil propagar toda esta información cuando el paso de tiempo es demasiado largo. Cuando una red tiene demasiadas capas profundas, se vuelve inentrenable. Este problema se llama: **problema de degradado de desaparición**. Si recuerda, la red neuronal actualiza el peso usando el algoritmo de descenso de gradiente. Los degradados se vuelven más pequeños cuando la red avanza hacia abajo a capas más bajas.

En conclusión, los gradientes permanecen constantes, lo que significa que no hay espacio para la mejora. El modelo aprende de un cambio en el degradado; este cambio afecta a la salida de la red. Sin embargo, si la diferencia en el gradiente es demasiado pequeña (es decir, los pesos cambian un poco), la red no puede aprender nada y así la salida. Por lo tanto, una red que enfrenta un problema de gradiente de desaparición no puede converger hacia una buena solución.

## Mejora LSMT

Para superar el potencial problema del gradiente de desaparición que enfrenta RNN, tres investigadores, Hochreiter, Schmidhuber y Bengio mejoraron el RNN con una arquitectura llamada Memoria Larga a Corto Plazo (LSTM). En resumen, LSTM proporciona a la red información relevante pasada a tiempo más reciente. La máquina utiliza una mejor arquitectura para seleccionar y llevar información a tiempo posterior.

La arquitectura LSTM está disponible en TensorFlow, `tf.contrib.rnn.lstmcell`. LSTM está fuera del alcance del tutorial. Puede consultar la documentación oficial para más información

## RNN en series temporales

En este tutorial, va a utilizar un RNN con datos de series de tiempo. Las series temporales dependen del tiempo anterior, lo que significa que los valores pasados incluyen información relevante de la que la red puede aprender. La idea detrás de la predicción de series temporales es estimar el valor futuro de una serie, digamos, el precio de las acciones, la temperatura, el PIB, etc.

La preparación de datos para RNN y series temporales puede ser un poco complicada. En primer lugar, el objetivo es predecir el siguiente valor de

la serie, lo que significa que utilizará la información pasada para estimar el valor en  $t + 1$ . La etiqueta es igual a la secuencia de entrada y se desplazó un punto por delante. En segundo lugar, el número de entrada se establece en 1, es decir, una observación por vez. Por último, el paso de tiempo es igual a la secuencia del valor numérico. Por ejemplo, si establece el paso de tiempo en 10, la secuencia de entrada devolverá diez veces consecutivas.

Mira el gráfico de abajo, hemos representado los datos de la serie temporal a la izquierda y una secuencia de entrada ficticia a la derecha. Crear una función para devolver un conjunto de datos con valor aleatorio para cada día de enero de 2001 a diciembre de 2016

```
To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
def create_ts(start = '2001', n = 201, freq = 'M'):
 rng = pd.date_range(start=start, periods=n, freq=freq)
 ts = pd.Series(np.random.uniform(-18, 18, size=len(rng)),
rng).cumsum()
 return ts
ts= create_ts(start = '2001', n = 192, freq = 'M')
ts.tail(5)
```

## Salida

```
2016-08-31 -93.459631
2016-09-30 -95.264791
2016-10-31 -95.551935
2016-11-30 -105.879611
2016-12-31 -123.729319
Freq: M, dtype: float64
```

```
ts = create_ts(start = '2001', n = 222)

Left
plt.figure(figsize=(11,4))
plt.subplot(121)
```

```

plt.plot(ts.index, ts)
plt.plot(ts.index[90:100], ts[90:100], "b-", linewidth=3, label="A
training instance")
plt.title("A time series (generated)", fontsize=14)

Right
plt.subplot(122)
plt.title("A training instance", fontsize=14)
plt.plot(ts.index[90:100], ts[90:100], "b-", markersize=8,
label="instance")
plt.plot(ts.index[91:101], ts[91:101], "bo", markersize=10,
label="target", markerfacecolor='red')
plt.legend(loc="upper left")
plt.xlabel("Time")

plt.show()

```



La parte derecha del gráfico muestra todas las series. Comienza desde 2001 y termina en 2019. No tiene sentido alimentar todos los datos en la red, en su lugar, debe crear un lote de datos con una longitud igual al paso de tiempo. Este lote será la variable X. La variable Y es la misma que X pero desplazada por un punto (es decir, desea pronosticar  $t + 1$ ).

Ambos vectores tienen la misma longitud. Puede verlo en la parte derecha del gráfico anterior. La línea representa los diez valores de la

entrada X, mientras que los puntos rojos son los diez valores de la etiqueta, Y. Tenga en cuenta que, la etiqueta comienza un período por delante de X y termina un período después.

# Construye un RNN para predecir series temporales en TensorFlow

Ahora, es hora de construir su primer RNN para predecir la serie anterior. Debe especificar algunos hiperparámetros (los parámetros del modelo, es decir, número de neuronas, etc.) para el modelo:

- Número de entrada: 1
- Paso de tiempo (ventanas en series de tiempo): 10
- Número de neuronas: 120
- Número de salida: 1

Su red aprenderá de una secuencia de 10 días y contendrá 120 neuronas recurrentes. Alimenta el modelo con una entrada, es decir, un día. No dude en cambiar los valores para ver si el modelo ha mejorado.

Antes de construir el modelo, debe dividir el dataset en un conjunto de trenes y un conjunto de pruebas. El conjunto de datos completo tiene 222 puntos de datos; utilizará los primeros 201 puntos para entrenar el modelo y los últimos 21 puntos para probar su modelo.

Después de definir un conjunto de tren y pruebas, debe crear un objeto que contenga los lotes. En estos lotes, tiene valores X e valores Y. Recuerde que los valores de X están retrasados con un período. Por lo tanto, se utilizan las primeras 200 observaciones y el paso de tiempo es igual a 10. El objeto X\_batch debe contener 20 lotes de tamaño  $10 * 1$ . Los y\_lotes tienen la misma forma que el objeto X\_lotes pero con un punto por delante.

## Paso 1) Crear el tren y probar

En primer lugar, convierte la serie en una matriz numpy; a continuación,

define las ventanas (es decir, el número de tiempo de que la red aprenderá), el número de entrada, salida y el tamaño del conjunto de trenes.

```
series = np.array(ts)
n_windows = 20
n_input = 1
n_output = 1
size_train = 201
```

Después de eso, simplemente divide la matriz en dos datasets.

```
Split data
train = series[:size_train]
test = series[size_train:]
print(train.shape, test.shape)
(201,) (21,)
```

## Paso 2) Cree la función para devolver X\_batches y y\_batches

Para hacerlo más fácil, puede crear una función que devuelve dos matrices diferentes, una para X\_lotes y otra para y\_lotes.

Vamos a escribir una función para construir los lotes.

Tenga en cuenta que, los X lotes se retrasan en un punto (tomamos el valor t-1). La salida de la función debe tener tres dimensiones. Las primeras dimensiones son iguales al número de lotes, la segunda es el tamaño de las ventanas y la última es el número de entrada.

La parte difícil es seleccionar los puntos de datos correctamente. Para los puntos de datos X, elija las observaciones de t = 1 a t = 200, mientras que para el punto de datos Y, devuelven las observaciones de t = 2 a 201. Una vez que tenga los puntos de datos correctos, es sencillo cambiar la forma de la serie.

Para construir el objeto con los lotes, debe dividir el dataset en diez lotes

de igual longitud (es decir, 20). Puede utilizar el método de reconfiguración y pasar -1 para que la serie sea similar al tamaño del lote. El valor 20 es el número de observaciones por lote y 1 es el número de entrada.

Debe hacer el mismo paso, pero para la etiqueta.

Tenga en cuenta que, debe cambiar los datos al número de tiempo que desea pronosticar. Por ejemplo, si desea predecir un tiempo de anticipación, entonces cambia la serie por 1. Si desea pronosticar dos días, cambie los datos por 2.

```
x_data = train[:size_train-1]: Select all the training instance
minus one day
X_batches = x_data.reshape(-1, windows, input): create the right
shape for the batch e.g (10, 20, 1)
def create_batches(df, windows, input, output):
 ## Create X
 x_data = train[:size_train-1] # Select the data
 X_batches = x_data.reshape(-1, windows, input) # Reshape
the data
 ## Create y
 y_data = train[n_output:size_train]
 y_batches = y_data.reshape(-1, windows, output)
 return X_batches, y_batches
```

Ahora que la función está definida, puede llamarla para crear los lotes.

```
X_batches, y_batches = create_batches(df = train,
 windows = n_windows,
 input = n_input,
 output = n_output)
```

Puede imprimir la forma para asegurarse de que las dimensiones son correctas.

```
print(X_batches.shape, y_batches.shape)
(10, 20, 1) (10, 20, 1)
```

Debe crear el conjunto de pruebas con solo un lote de datos y 20 observaciones.

Tenga en cuenta que, si pronostica días después de días, significa que el segundo valor previsto se basará en el valor verdadero del primer día ( $t + 1$ ) del conjunto de datos de prueba. De hecho, se conocerá el verdadero valor.

Si desea pronosticar  $t + 2$  (es decir, dos días antes), debe usar el valor predicho  $t + 1$ ; si va a predecir  $t + 3$  (tres días antes), debe usar el valor predicho  $t + 1$  y  $t + 2$ . Tiene sentido que, es difícil predecir con precisión  $t + n$  días por delante.

```
x_test, y_test = create_batches(df = test, windows = 20, input = 1,
output = 1)
print(x_test.shape, y_test.shape)
(10, 20, 1) (10, 20, 1)
```

Muy bien, su tamaño de lote está listo, puede construir la arquitectura RNN. Recuerda, tienes 120 neuronas recurrentes.

### **Paso 3)** Construir el modelo

Para crear el modelo, debe definir tres partes:

1. La variable con los tensores
2. El RNN
3. La pérdida y optimización

#### **Paso 3.1)** Variables

Debe especificar las variables X e y con la forma adecuada. Este paso es trivial. El tensor tiene la misma dimensión que los objetos X\_lotes y y\_lotes.

Por ejemplo, el tensor X es un marcador de posición (Compruebe el

tutorial sobre Introducción a Tensorflow para refrescar su mente sobre la declaración de variables) tiene tres dimensiones:

- Nota: tamaño del lote
- n\_windows: Longitud de las ventanas. Es decir, el número de veces que el modelo mira hacia atrás
- n\_input: Número de entrada

El resultado es:

```
tf.placeholder(tf.float32, [None, n_windows, n_input])
```

```
1. Construct the tensors
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])
```

### Paso 3.2) Crear el RNN

En la segunda parte, debe definir la arquitectura de la red. Como antes, se utiliza el objeto BasicRNNCell y dynamic\_rnn desde el estimador TensorFlow.

```
2. create the model
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, x,
dtype=tf.float32)
```

La siguiente parte es un poco más complicada, pero permite un cálculo más rápido. Debe transformar la salida de ejecución en una capa densa y luego convertirla de nuevo para tener la misma dimensión que la entrada.

```
stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

### Paso 3.3) Crear la pérdida y la optimización

La optimización del modelo depende de la tarea que esté realizando. En el tutorial anterior sobre CNN, su objetivo era clasificar imágenes, en este tutorial, el objetivo es ligeramente diferente. Se le pide que haga una predicción sobre una variable continua en comparación con una clase.

Esta diferencia es importante porque cambiará el problema de optimización. El problema de optimización para una variable continua es minimizar el error cuadrado medio. Para construir estas métricas en TF, puede usar:

- `tf.reduce_sum(tf.square(outputs - y))`

El resto del código es el mismo que antes; usa un optimizador Adam para reducir la pérdida (es decir, MSE):

- `tf.train.AdamOptimizer(learning_rate=learning_rate)`
- `optimizer.minimize(loss)`

Eso es todo, puedes empacar todo junto, y tu modelo está listo para entrenar.

```
tf.reset_default_graph()
r_neuron = 120

1. Construct the tensors
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])

2. create the model
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X,
dtype=tf.float32)

stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

```

3. Loss + optimization
learning_rate = 0.001

loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()

```

Entrenará el modelo usando 1500 épocas e imprimirá la pérdida cada 150 iteraciones. Una vez entrenado el modelo, evaluará el modelo en el conjunto de pruebas y creará un objeto que contenga las predicciones.

```

iteration = 1500

with tf.Session() as sess:
 init.run()
 for iters in range(iteration):
 sess.run(training_op, feed_dict={X: X_batches, y:
y_batches})
 if iters % 150 == 0:
 mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
 print(iters, "\tMSE:", mse)

 y_pred = sess.run(outputs, feed_dict={X: X_test})
0 MSE: 502893.34
150 MSE: 13839.129
300 MSE: 3964.835
450 MSE: 2619.885
600 MSE: 2418.772
750 MSE: 2110.5923
900 MSE: 1887.9644
1050 MSE: 1747.1377
1200 MSE: 1556.3398
1350 MSE: 1384.6113

```

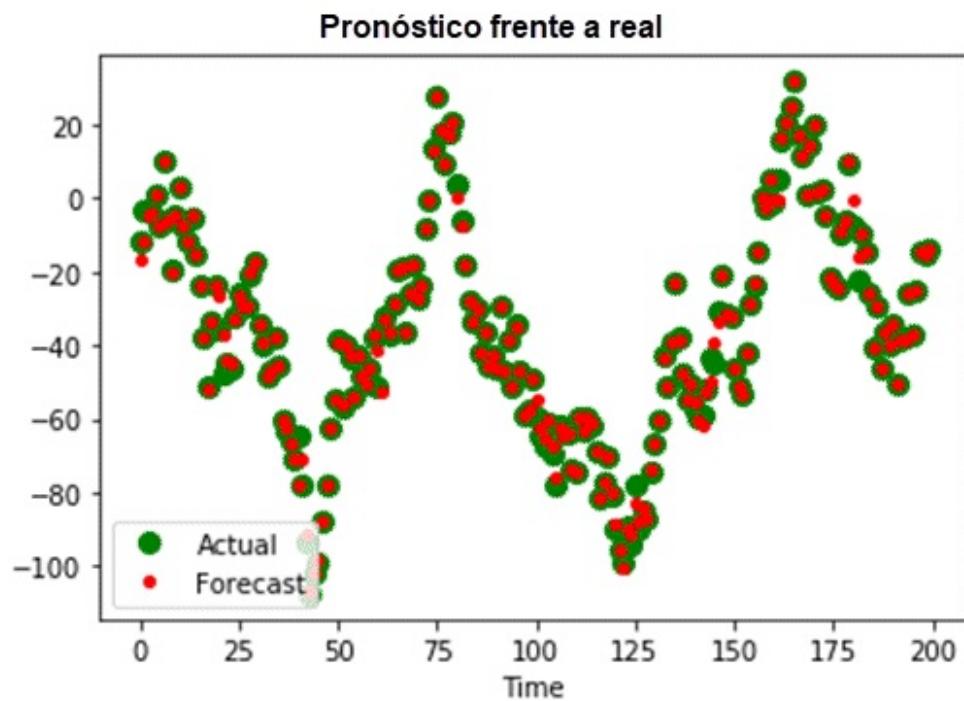
Por fin, puede trazar el valor real de la serie con el valor previsto. Si se corrige el modelo, los valores previstos deben colocarse encima de los valores reales.

Como puede ver, el modelo tiene un margen de mejora. Depende de usted

cambiar los hiperparámetros como las ventanas, el tamaño del lote o el número de neuronas recurrentes.

```
plt.title("Forecast vs Actual", fontsize=14)
plt.plot(pd.Series(np.ravel(y_test)), "bo", markersize=8,
label="Actual", color='green')
plt.plot(pd.Series(np.ravel(y_pred)), "r.", markersize=8,
label="Forecast", color='red')
plt.legend(loc="lower left")
plt.xlabel("Time")

plt.show()
```



## Resumen

Una red neuronal recurrente es una arquitectura robusta para tratar series temporales o análisis de texto. La salida del estado anterior es retroalimentación para preservar la memoria de la red a lo largo del tiempo o secuencia de palabras.

En TensorFlow, puede utilizar los siguientes códigos para entrenar una

red neuronal recurrente para series de tiempo:

## Parámetros del modelo

```
n_windows = 20
n_input = 1
n_output = 1
size_train = 201
```

## Definir el modelo

```
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron,
activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X,
dtype=tf.float32)

stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

## Construir la optimización

```
learning_rate = 0.001

loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

## Entrenar el modelo

```
init = tf.global_variables_initializer()
iteration = 1500

with tf.Session() as sess:
 init.run()
 for iters in range(iteration):
 sess.run(training_op, feed_dict={X: X_batches, y:
y_batches})
 if iters % 150 == 0:
```

```
 mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
 print(iters, "\tMSE:", mse)

y_pred = sess.run(outputs, feed_dict={X: X_test})
```