008. TENSORFLOW COMO ALTERNATIVA A HERRAMIENTAS ESTADISTICAS, CASO DE APLICACIÓN: REGRESIÓN LINEAL

Autor:

Luis Enrique Espinoza Mendoza
Universidad Ecotec
Docente Tiempo Completo
Maestría en Docencia y Currículo
lespinoza@ecotec.edu.ec

Resumen

En la actualidad, con la llegada de conocimientos o términos nuevos informáticos, como son el aprendizaje automático, inteligencia de negocios, inteligencia artificial, softwares y lenguajes de programación, tanto generales como específicos, están incorporando en sus librerías, funciones matemáticas, estadísticas, de cálculo, ya que, como en todo fenómeno de revolución tecnológica, siempre se presenta una etapa en la cual, no se determina aún un camino a seguir.

En lo que se refiere a software específico estadístico, se tiene a R, que en estos momentos lidera sobre los demás, pero es un lenguaje muy específico, es decir, para otras funcionalidades fuera de estadística, no es de mucha utilidad.

Este inconveniente, se presenta en los nuevos lenguajes, es decir, Python, R, F#, pueden brindarnos mucha ayuda en tareas puntuales, pero en la práctica, hay que usar varios al mismo tiempo, trabajando entre sí, y esto también depende, del escenario al que se apliquen.

El presente artículo, aborda uno de estos lenguajes, TensorFlow, el cual se lo clasifica como un lenguaje para aprendizaje automático, y como tal, tiene un fuerte componente matemático.

A través, de una aplicación matemática, como es la regresión lineal, se intenta demostrar las bondades del lenguaje, ya que, tiene un crecimiento menos desesperado, que las otras herramientas, y, desde su concepción, presenta un roadmap, más estable y pensado.

PALABRAS CLAVE:TensorFlow, Regresión Lineal, Aprendizaje Automático

INTRODUCCION:

Como objetivo principal de este artículo, se desea demostrar las bondades matemáticas que presenta TensorFlow, particularmente para su aplicación en estadística.

Este planteamiento, surge, debido a que, así como han evolucionado los otros lenguajes, ahora se abre un mundo nuevo en cuanto a tecnología, como lo son las derivadas del aprendizaje automático, y el entendimiento de este marco de trabajo, para su aplicación en los negocios y en la investigación.

El caballo de batalla del aprendizaje automático, son las redes neuronales, y es en punto que los otros lenguajes como R, F#, o Python, se quedan cortos.

Pero del mismo modo, se necesita analizar o desglosar los datos generados por las redes neuronales, y en esto, software especializado para redes neuronales, se queda corto, por lo cual, ahí se debe incorporar a los softwares estadísticos.

Con esto en mente, se muestra una opción, que se puede desempeñar bien en ambos mundos, como lo es TensorFlow, el cual tiene el respaldo de Google, que es la empresa que lo creo.

El motivo por el cual se tomó de todas las operaciones matemáticas, la regresión lineal, es que, este modelo, es el más básico en lo que se refiere a algoritmos predictivos de datos, y más fácil de comparar de entre los demás algoritmos predictivos.

Entre los resultados que se encontró, es que la versión de api más adecuada para usar, es la que trabaja sobre Python, ya que esta mejor documentada.

También, se pudo determinar que las posibilidades de cálculo en cuanto a capacidad de carga de datos son mucho más interesantes que cualquier otro lenguaje, ya que, las librerías de TensorFlow están optimizados para trabajar con el GPU del computador, lo cual es un aspecto innovador, con lo cual, cálculos que podrían ser muy complicados o restrictivos por el tipo de hardware necesario, son posibles, gracias a que TensorFlow puede utilizar además del CPU, el GPU.

DESARROLLO:

MARCO TEORICO

Redes de neuronas
Existen varias maneras de implementar aprendizaje profundo, una de las más comunes es utilizar redes de neuronas. Una red de neuronas es una herramienta matemática que modela, de forma muy simplificada, el funcionamiento de las neuronas en el cerebro.

Es una serie de operaciones matemáticas sobre una lista de números, que da como resultado otra lista de números. Otra forma de verlas, es como un procesador de información, que recibe información entrante, codificada como números, procesa la data, y produce como resultado información saliente, codificada como otros números.

Una aplicación sería una red de neuronas que detecte rostros en imágenes, donde se codifica una imagen como una lista de números.

En este ejemplo, esta red neuronal, recibiría tantos números a su entrada como píxeles tienen las imágenes. Si la información que se espera a la salida es que se diga si hay un rostro o no, basta con un solo número en la lista saliente.

Si el número que sale de la red, toma un valor cercano a 1.0 significa que hay un rostro, y si toma un valor cercano a 0.0 significa que no lo hay. Valores intermedios se pueden interpretar como inseguridad, o probabilidad.

Arquitectura
En el siguiente diagrama se observa la arquitectura de una red de neuronas. Cada círculo representa una neurona. Las neuronas se organizan en capas, de la siguiente forma: las neuronas amarillas son las entradas, y reciben cada uno de los números de la lista de números entrante, las neuronas verdes son las salidas, y una vez que la red realiza su operación matemática, contienen el resultado, también como una lista de números; las neuronas celestes son neuronas ocultas, que contienen cálculos intermedios de la red.

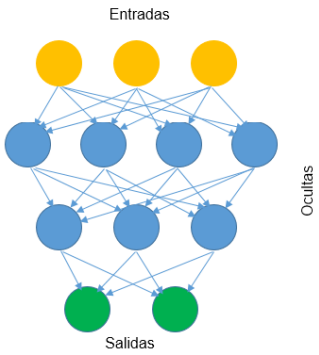


Figura 1: Arquitectura Red Neuronal. Propia Autoría.

Normalmente todas las neuronas de cada capa tienen una conexión con cada neurona de la siguiente capa, como se representa en el diagrama. Estas conexiones

tienen asociado un número, que se llama peso. La principal operación que realiza la red de neuronas consiste en multiplicar los valores de una neurona por los pesos de sus conexiones salientes. Cada neurona de la siguiente capa recibe números de varias conexiones entrantes, y lo primero que hace es sumarlos todos.

Función de activación
Hay otra operación que realizan todas las capas salvo la capa de entrada, antes de continuar multiplicando sus valores por las conexiones salientes, se trata de la función de activación.

Esta función recibe como entrada la suma de todos los números que llegan por las conexiones entrantes, transforma el valor mediante una fórmula, y produce un nuevo número. Existen varias opciones, pero una de las funciones más habituales es la función sigmoide. Uno de los objetivos de la función de activación es mantener los números producidos por cada neurona dentro de un rango razonable (por ejemplo, números reales entre 0 y 1).

La función *sigmoide* es no lineal, esto significa que, si dibujamos en una gráfica los valores de entrada en un eje y los de salida en el otro eje, el dibujo no será una línea. Esto es muy importante, porque si la función de activación que elegimos es lineal, la red estará limitada a resolver problemas lineales.

Un ejemplo de problema lineal sería la conversión de temperatura entre Celsius (Europa) y Fahrenheit (EEUU). Si se dibuja uno frente a otro en una gráfica, el resultado será una línea.

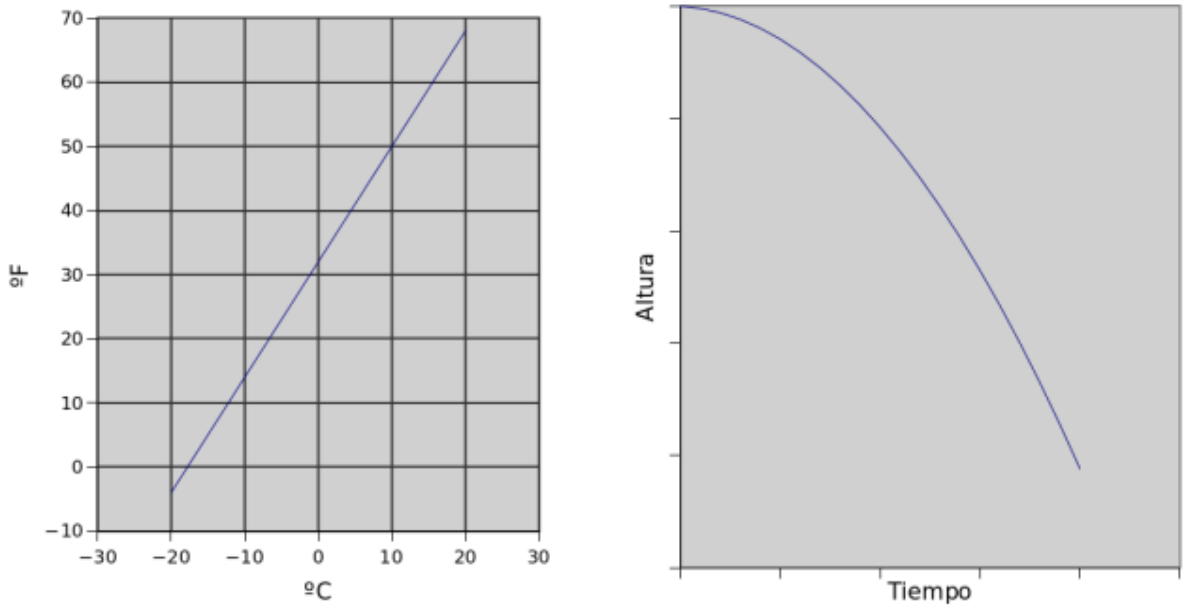


Figura 2: Transformación Lineal vs No Lineal. Tomado http://www.sc.ehu.es/sbweb/fisica3/datos/ajuste/nolineal.html

Transformación lineal (izquierda) frente a no lineal (derecha).

Implementación

Una forma sencilla de implementar redes de neuronas consiste en almacenar los pesos en matrices. Si se guarda los valores de todas las neuronas de una capa en un vector, el producto del vector y la matriz de pesos de salida, da los valores de entrada de cada neurona en la siguiente capa.

Ahora sólo falta aplicar la función de activación elegida a cada elemento de ese segundo vector, y repetir el proceso. Si se dispone de una biblioteca que implemente matrices, se puede implementar la red en unas pocas líneas.

Aprendizaje en redes de neuronas

Suponiendo que se tiene claro cuantas neuronas hacen falta a la entrada y a la salida, porque ya se ha decidido cómo representar nuestra información en forma de listas de números. Todavía faltan varias cosas por decidir para tener una red que funcione:

- ¿Cuántas capas ocultas se incluirán?
- ¿Cuántas neuronas se colocarán en cada capa oculta?
- ¿Qué pesos concretos se usará en las conexiones entre cada par de capas?

Habitualmente los dos primeros puntos se deciden a mano, y algunas veces mediante prueba y error. Lógicamente, cuantas más neuronas se tiene en capas ocultas, más compleja es la red, y podrá resolver a su vez problemas más complejos. Por otra parte, cuantas más neuronas ocultas se tiene, más costará realizar todos los productos y sumas.

Si se elige una función de activación lineal, no merece la pena utilizar capas ocultas en absoluto, porque se puede comprobar que la potencia de la red será la misma independiente de la cantidad de capas. La potencia de la red sólo aumenta con el número de capas para funciones de activación no lineales, como la *sigmoide*.

El tercer punto, se puede resolver de forma automática, mediante un proceso llamado entrenamiento. Para entrenar una red de neuronas, se necesita primero recopilar algunos ejemplos de entradas y la salida que se desea para cada ejemplo.

Por ejemplo, en el detector de rostros, se necesita necesitamos recopilar ejemplos de imágenes con rostros, y ejemplos de imágenes sin rostros. También se necesita etiquetar cuales son las imágenes que tienen rostros y cuales las que no, porque esa es precisamente la salida deseada para cada ejemplo. Este proceso de entrenamiento se conoce como aprendizaje supervisado, porque el sistema necesita de un supervisor que le explique lo que tiene que hacer (mediante ejemplos de entradas y salidas).

Aprendizaje Profundo o Deep Learning

En la actualidad, las empresas grandes necesitan cada vez mejorar sus procesos, tomar decisiones más rápidas y correctas. Este fenómeno, logró que aparecieran términos nuevos como son la inteligencia de negocios y el minado de datos. Pero, ya de esto, varios años.

Pero desde esa época, hacia acá, ha crecido aún más la cantidad de datos que se generan, motivos por el cual, así como evoluciono el software, hablando de los sistemas operativos y bases de datos, tanto verticales y horizontales, también lo hicieron las técnicas y algoritmos empleados para el análisis de los datos.

En estos procesos de vanguardia, que ya se viven, hace su aparición el aprendizaje automático o machine learning, donde, se desea que los análisis que son repetitivos, sean aprendidos por el computador, y de esta manera, aumenta la complejidad de los mismos.

En estos algoritmos de aprendizaje automático, a su vez, los mismos, también crecieron y especializaron, donde ahora se habla del Deep learning o aprendizaje profundo, donde se desea responder preguntas complejas y construir sistemas inteligentes.

El aprendizaje profundo, se utiliza hoy para comprender el contenido de las imágenes, el lenguaje natural y el habla, en sistemas que van desde aplicaciones móviles a vehículos autónomos. Los desarrollos en este campo están teniendo lugar a una velocidad vertiginosa, y, los mismos se extienden a otros dominios y tipos de datos, como estructuras químicas y genéticas complejas para el descubrimiento de fármacos y registros médicos de alta calidad en la salud pública.

Los métodos de aprendizaje profundo, que también reciben el nombre de redes neuronales profundas, eran originalmente inspirados en la vasta red de neuronas interconectadas del cerebro humano. En aprendizaje profundo, se alimenta millones de instancias de datos en una red de neuronas, enseñándoles a reconocer patrones de entradas crudas.

Las redes neuronales profundas, toman entradas sin procesar (como valores de píxeles en una imagen) y las transforman en representaciones útiles, extrayendo características de mayor nivel (como formas y bordes en imágenes) que capturan conceptos complejos mediante la combinación de piezas de información cada vez más pequeñas para resolver tareas desafiantes como la clasificación de imágenes.

Las redes aprenden automáticamente a construir representaciones abstractas, adaptándose y corrigiéndose a sí mismas, ajustando los patrones observados en los datos. La capacidad de construir automáticamente representaciones de datos, es una ventaja clave de redes neuronales profundas sobre convencionales, que generalmente requiere experiencia en el dominio y funciones manuales de ingeniería antes de que cualquier "aprendizaje" pueda ocurrir.

TensorFlow
Son librerías de código abierto, donde, este marco de trabajo opera sobre lo que constituye aprendizaje profundo.

Este tipo de algoritmos, se han utilizado por varios años en muchos productos y áreas en Google, como lo son búsqueda, traducción, publicidad, visión artificial y reconocimiento de voz e imágenes.

TensorFlow es un sistema de segunda generación, para implementar y desplegar redes neuronales. Fue lanzado al público como un marco de código abierto con Apache 2.0, en noviembre del 2015, donde en sus primeros inicios, se dedicó a proyectos internos de Google, pero, gracias a su estabilidad y flexibilidad, combinado con la experiencia de los Ingenieros de Google, que son el motor que da manteamiento y que continúa desarrollando las librerías, han convertido a TensorFlow en un sistema líder para hacer aprendizaje profundo.

Un área principal, donde el aprendizaje profundo es sumamente útil, es el reconocimiento y gestión de imágenes por computadora.

Como tarea principal en estas funciones, es la construcción de algoritmos de clasificación de imágenes, que reciben imágenes como entrada, y devuelven un conjunto de categorías que las describen a ellas.

Investigadores, científicos de datos e ingenieros han diseñado redes neuronales profundas, las cuales obtienen resultados altamente precisos en la comprensión del contenido visual.

Estas redes profundas, son típicamente entrenadas con una cantidad de imágenes muy grande, tomando mucho tiempo, recursos y esfuerzo. Sin embargo, en una tendencia creciente, los investigadores están lanzados modelos de redes neuronales previamente entrenados, que otros usuarios pueden descargar y aplicar a sus datos.

Regresión Lineal

El análisis de regresión lineal es una técnica estadística utilizada para estudiar la relación entre variables. Se adapta a una amplia variedad de situaciones. En la investigación social, el análisis de regresión se utiliza para predecir un amplio rango de fenómenos, desde medidas económicas hasta diferentes aspectos del comportamiento humano.

En el contexto de la investigación de mercados puede utilizarse para determinar en cuál de diferentes medios de comunicación puede resultar más eficaz invertir; o para predecir el número de ventas de un determinado producto.

En física se utiliza para caracterizar la relación entre variables o para calibrar medidas. Tanto en el caso de dos variables (regresión simple) como en el de más de dos variables (regresión múltiple), el análisis de regresión lineal puede utilizarse para explorar y cuantificar la relación entre una variable llamada dependiente o criterio (Y) y una o más variables llamadas independientes o predictoras (X_1, X_2, ..., X_k), así como para desarrollar una ecuación lineal con fines predictivos.

Además, el análisis de regresión lleva asociados una serie de procedimientos de diagnóstico (análisis de los residuos, puntos de influencia) que informan sobre la estabilidad e idoneidad del análisis y que proporcionan pistas sobre cómo perfeccionarlo.

La recta de regresión

Un diagrama de dispersión ofrece una idea bastante aproximada sobre el tipo de relación existente entre dos variables. Pero, además, un diagrama de dispersión

también puede utilizarse como una forma de cuantificar el grado de relación lineal existente entre dos variables: basta con observar el grado en el que la nube de puntos se ajusta a una línea recta.
Ahora bien, aunque un diagrama de dispersión permite formarse una primera impresión muy rápida sobre el tipo de relación existente entre dos variables, utilizarlo como una forma de cuantificar esa relación tiene un serio inconveniente: la relación entre dos variables no siempre es perfecta o nula; de hecho, habitualmente no es ni lo uno ni lo otro.

Supóngase que se dispone de un pequeño conjunto de datos con información sobre 35 marcas de cerveza y que estamos interesados en estudiar la relación entre el grado de alcohol de las cervezas y su contenido calórico.

Un buen punto de partida para tener una primera impresión de esa relación podría ser la representación de la nube de puntos, tal como muestra el diagrama de dispersión:

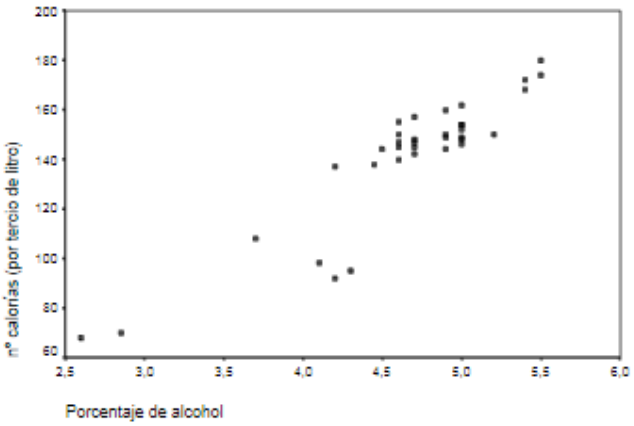


Figura 3: Gráfico de Dispersión de % de Alcohol vs No. Calorías. Generado por TensorFlow

El eje vertical muestra el número de calorías (por cada tercio de litro) y el horizontal el contenido de alcohol (expresado en porcentaje). A simple vista, parece existir una relación positiva entre ambas variables: conforme aumenta el porcentaje de alcohol, también aumenta el número de calorías.

En esta muestra no hay cervezas que teniendo alto contenido de alcohol tengan pocas calorías y tampoco hay cervezas que teniendo muchas calorías tengan poco alcohol.

La mayor parte de las cervezas de la muestra se agrupan entre el 4,5 % y el 5 % de alcohol, siendo relativamente pocas las cervezas que tienen un contenido de alcohol inferior a ése. Se podría haber extendido el rango de la muestra incluyendo cervezas sin alcohol, pero el rango de calorías y alcohol considerados parece bastante apropiado: no hay, por ejemplo, cervezas con un contenido de alcohol del 50 %, o cervezas sin calorías.

Al describir los datos, se puede decir simplemente que el aumento del porcentaje de alcohol va acompañado de un aumento en el número de calorías; pero esto, aunque correcto, es poco específico.

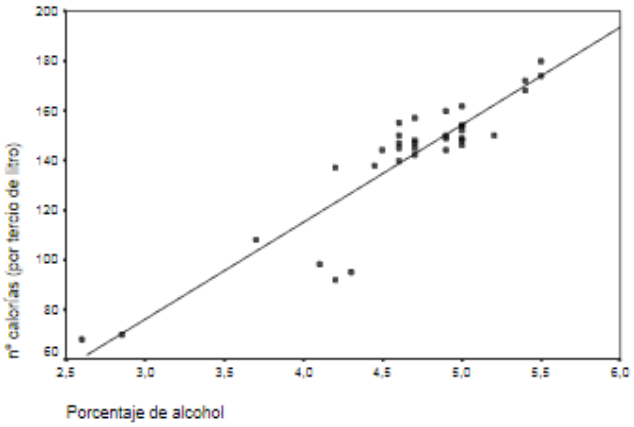
Se puede describir la pauta observada en la nube de puntos mediante una función matemática simple, tal como una línea recta. A primera vista, una línea recta podría ser un buen punto de partida para describir resumidamente la nube de puntos de la figura.

Puesto que una línea recta posee una fórmula muy simple, $Y_i = B_0 + B_1 X_i$, se puede comenzar obteniendo los coeficientes B_0 y B_1 que definen la recta.

El coeficiente B_1 es la pendiente de la recta: el cambio medio que se produce en el número de calorías (Y_i) por cada unidad de cambio que se produce en el porcentaje de alcohol (X_i).

El coeficiente B_0 es el punto en el que la recta corta el eje vertical: el número medio de calorías que corresponde a una cerveza con porcentaje de alcohol cero.

Al conocer los valores de estos dos coeficientes, se puede reproducir la recta y describir con ella la relación existente entre el contenido de alcohol y el número de calorías.



$$Y_i = -33{,}77 + 37{,}65\, X_i$$

nº de calorías = − 33,77 + 37,65 (% de alcohol)

Figura 4: Recta ajustada a Gráfico de Dispersión de % de Alcohol vs No. Calorías. Generado por TensorFlow

En general, la recta hace un seguimiento bastante bueno de los datos. La fórmula de la recta aparece a la derecha del diagrama. La pendiente de la recta (B_1) indica que, en promedio, a cada incremento de una unidad en el porcentaje de alcohol (X_i) le corresponde un incremento de 37,65 calorías (Y_i). El origen de la recta (B_0) sugiere que una cerveza sin alcohol (grado de alcohol cero) podría contener 33,77 calorías. Y esto, obviamente, no parece posible.

Al examinar la nube de puntos se ve que la muestra no contiene cervezas con menos de un 2 % de alcohol. Así, aunque el origen de la recta aporta información sobre lo que podría ocurrir si se extrapola hacia abajo la pauta observada en los datos hasta llegar a una cerveza con grado de alcohol cero, al hacer esto se estaría efectuando pronósticos en un rango de valores que va más allá de lo que abarcan los datos disponibles, y eso es algo extremadamente arriesgado en el contexto del análisis de regresión

La mejor recta de regresión

En una situación ideal (e irreal) en la que todos los puntos de un diagrama de dispersión se encontraran en una línea recta, no habría que preocuparse de encontrar la recta que mejor resume los puntos del diagrama.

Simplemente uniendo los puntos entre sí se obtendría la recta con mejor ajuste a la nube de puntos. Pero en una nube de puntos más realista es posible trazar muchas rectas diferentes. Obviamente, no todas ellas se ajustarán igualmente bien a la nube de puntos. Se trata de encontrar la recta capaz de convertirse en el mejor representante del conjunto total de puntos.

Existen diferentes procedimientos para ajustar una función simple, cada uno de los cuales intenta minimizar una medida diferente del grado de ajuste.

La elección preferida ha sido, tradicionalmente, la recta que hace mínima la suma de los cuadrados de las distancias verticales entre cada punto y la recta.

Esto significa que, de todas las rectas posibles, existe una y sólo una que consigue que las distancias verticales entre cada punto y la recta sean mínimas (las distancias se elevan al cuadrado porque, de lo contrario, al ser unas positivas y otras negativas, se anularían unas con otras al sumarlas).

Bondad de ajuste

Además de acompañar la recta con su fórmula, podría resultar útil disponer de alguna indicación precisa del grado en el que la recta se ajusta a la nube de puntos. De hecho, la mejor recta posible no tiene por qué ser buena.

Se ha representado el porcentaje de alcohol de las cervezas (eje horizontal) y el precio de las mismas (eje vertical). Y no parece existir la misma pauta de asociación detectada entre las variables de la situación anterior.

Así pues, aunque siempre resulta posible, cualquiera que sea la nube de puntos, obtener la recta mínimo-cuadrática, se necesita información adicional para determinar el grado de fidelidad con que esa recta describe la pauta de relación existente en los datos.

Una medida de ajuste que ha recibido gran aceptación en el contexto del análisis de regresión es el coeficiente de determinación R^2: el cuadrado del coeficiente de correlación múltiple.

Se trata de una medida estandarizada que toma valores entre 0 y 1 (0 cuando las variables son independientes y 1 cuando entre ellas existe relación perfecta).

Este coeficiente posee una interpretación muy intuitiva: representa el grado de ganancia que se puede obtener al predecir una variable basándose en el conocimiento que se tiene de otra u otras variables.

Si se quiere, por ejemplo, pronosticar el número de calorías de una cerveza sin el conocimiento de otras variables, se utilizaría la media del número de calorías. Pero

si se tiene información sobre otra variable y del grado de relación entre ambas, es posible mejorar el pronóstico.

Metodología empleada, materiales y métodos
Para la aplicación de la regresión lineal con TensorFlow, el primer paso es crear datos los cuales serán usados en este modelo.

Se creó un archivo en Python, llamado regresión, a continuación, se muestra el código que permitió generar los datos:

```
import numpy as np
import matplotlib.pyplot as plt
x_train = np.linspace(-1, 1, 101)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33
plt.scatter(x_train, y_train)
plt.show()
```

En resumen, el código hace una llamada a la librería numpy, para poder tener acceso a las funciones matemáticas. También se incluye la librería matplotlib, para lograr la visualización de los datos.

Como datos de entrada, se ingresan 101 valores, entre -1 y 1, y la salida, es proporcional a los valores de entrada más un valor de ruido. A continuación, se muestran los puntos generados con la formula anterior:

En resumen, la función de regresión de mínimos cuadrados es:

```
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33
```

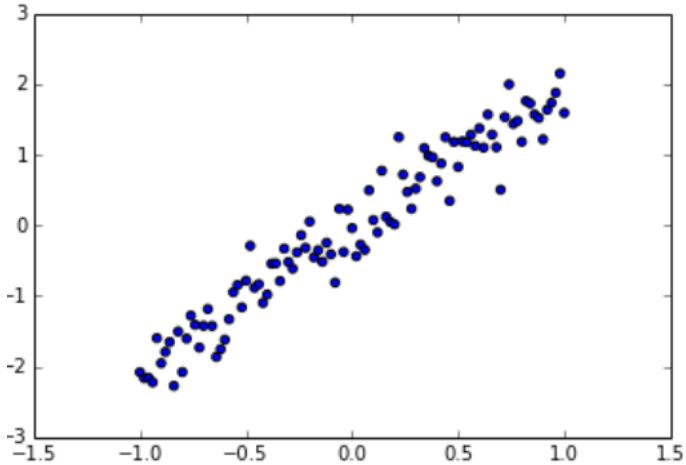


Figura 5: Gráfico de Dispersión de Resultados Obtenidos. Generado por TensorFlow

Ahora que se tiene algunos datos, se va a ajustar una línea, que ajuste de la mejor manera entre todos estos puntos.

Se debe proporciona a TensorFlow un puntaje para cada parámetro candidato que se analice.

Esta puntuación o peso se conoce comúnmente como una función de costo. Cuanto mayor sea el costo, peor será el parámetro del modelo.

Por ejemplo, si la línea que mejor se ajusta es y = 2x, una opción de parámetro de 2.01 debería tener un costo bajo, pero la elección de -1 debería tener un costo mayor.

TensorFlow se ocupa del funcionamiento interno e intenta actualizar los parámetros de forma eficiente para alcanzar eventualmente el mejor valor posible. Cada paso de recorrer todos sus datos para actualizar los parámetros se llama una época.

En este ejemplo, la manera de definir el costo, es como la suma de los errores. El error en predecir x, a menudo se calcula como la diferencia al cuadrado entre el valor actual de f(x) y el pronosticado.

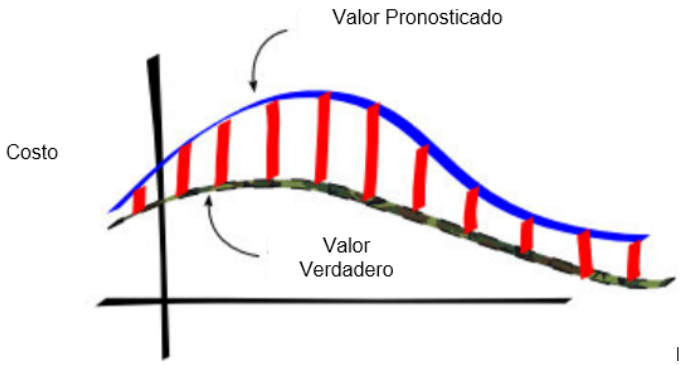


Figura 6: Representación de Valor Pronosticado vs Valor Verdadero. Copyright 2017, Machine Learning with TensorFow, Nishant Shukla

A continuación, se actualiza el código previo, donde se define la función de costo, y se pregunta a TensorFlow por un optimizador, para encontrar la solución óptima para los parámetros del modelo:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

learning_rate = 0.01
training_epochs = 100
x_train = np.linspace(-1, 1, 101)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
def model(X, w):
return tf.multiply(X, w)

w = tf.Variable(0.0, name="weights")
y_model = model(X, w)
cost = tf.square(Y-y_model)
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

```
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)

for epoch in range(training_epochs):
for (x, y) in zip(x_train, y_train):
sess.run(train_op, feed_dict={X: x, Y: y})
w_val = sess.run(w)
sess.close()
plt.scatter(x_train, y_train)
y_learned = x_train*w_val
plt.plot(x_train, y_learned, 'r')
plt.show()
```

En este código, se mantiene como modelo de regresión lineal:
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33,

Lo que se realiza es el ajuste u optimización de la curva, para esto, se define como el nuevo modelo o función:
y = w * X., donde W es el conjunto de variables de pesos definida, dada por:
w = tf.Variable(0.0, name="weights").

El costo se define como la diferencia entre el valor ideal y la respuesta del modelo. En este modelo presentado, el costo está calculado como la suma de los errores. La fórmula de costo incorporada es:
cost = tf.square(Y-y_model)

Como ultimo paso, se hace varias iteracciones entre los datos de entrenamientos, para tratar de minimizas los costos en cada pasada, al final, se vuelve a usar como datos de entrada al modelo los valores de X y Y actualizados con este procedimiento.:

```
for epoch in range(training_epochs):
for (x, y) in zip(x_train, y_train):
sess.run(train_op, feed_dict={X: x, Y: y})
```

En resumen, la resolución del modelo de regresión, es un resultado gráfico, no entrega un nuevo modelo, sino que, usando el modelo original de regresión, aplica una optimización por costos para mejorar la gráfica. Luego se define la operación que será llamada en cada iteración del algoritmo de aprendizaje. Se recorre cada ítem del conjunto de datos, para generar los nuevos valores f(x), y luego se trata de minimizar el costo.

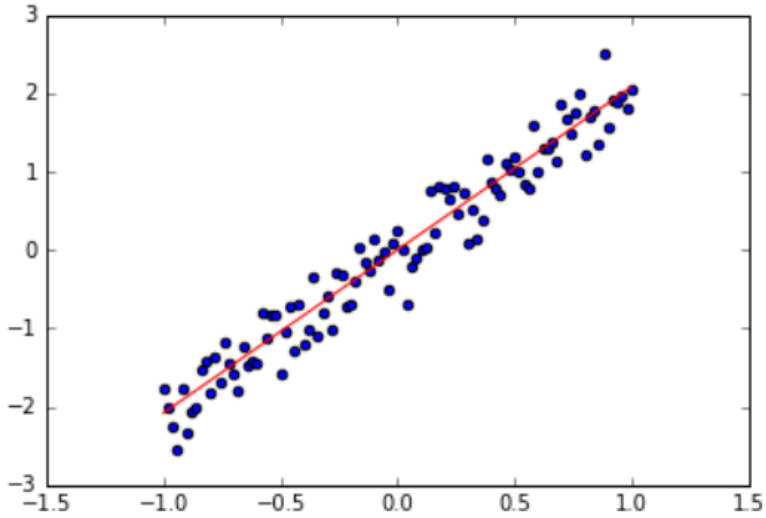


Figura 7: Recta de ajuste por mínimos cuadrados. Generado por TensorFlow

CONCLUSIONES:

TensorFlow es un framework poderoso, no sólo por la funcionalidad que presenta, sino, porque corre u opera sobre Python, entonces las posibilidades de acceso a funciones matemáticas son altísimas.

El artículo tiene como objetivo demostrar que TensorFlow, se puede tomar como otro lenguaje para uso estadístico. Al concluir este artículo, y revisar la funcionalidad existente, tiene puntos que son más robustos, porque la visión de estadística que posee este framework, si bien suple las necesidades básicas como se demostró con el ejercicio de regresión lineal, se debe considerar que las funciones matemáticas que posee son en sí más robustas, en cuanto posee muchos más algoritmos orientados a análisis predictivos, y a la creación de sistemas inteligentes.

La unión de TensorFlow intrínseca con Python, y a su vez, observando que Python se integra muy bien con R, forman una trilogía interesante, que es capaz de suplir todos los puntos de análisis estadístico a todos los niveles, sean básicos, intermedios o avanzados.

El otro punto novedoso en cuanto a TensorFlow, es su capacidad, de además de usar el CPU del computador, puede usar el GPU del mismo, lo cual aumente muchísimo el poder de computo. Al tener este aprovechamiento excepcional de hardware, algoritmos de clasificación como manejo de clusters, o procesamientos más avanzados, los cuales necesitaban de servidores o bases de datos distribuidas, son posibles en esta arquitectura, dándole un valor extra de mucho peso para los investigadores.

BIBLIOGRAFIA:

Ayyadevara, V. (2018). *Pro Machine Learning Algorithms.* Apress.

Bruce, A., & Bruce, P. (2017). *Practical Statistics for Data Scientist.* O'Reilly.

García, Á. F. (n.d.). *Curso Interactivo de Física en Internet*. Retrieved from Curso Interactivo de Física en Internet: http://www.sc.ehu.es/sbweb/fisica3/datos/ajuste/nolineal.html

Hope, T., Resheff, Y., & Lieder, I. (2017). *Learning TensorFlow.* O'Reilly. Retrieved from http://deeplearning.net/.

Lieder, I., Hope, T., & Resheff, Y. (2017). *Learning TensorFlow.* O'Reilly.

Lopez, R. (n.d.). *https://rubenlopezg.wordpress.com/*. Retrieved from https://rubenlopezg.wordpress.com/: https://rubenlopezg.wordpress.com/2014/05/07/que-es-y-como-funciona-deep-learning/

Michelucci, U. (2018). *Applied Deep Learning.* Apress.

Nandy , A., & Biswas, M. (2017). *Reinforcement Learning.* Apress.

Pattanayak, S. (2017). *Pro Deep Learning with TensorFlow.* Apress.

Ramsundar, B., & Bosagh Zadeh, R. (2018). *TensorFlow for Deep Learning.* O'Reilly.

Shukla, N. (2017). *Machine Learning with TensorFlow.* MEAP.

Standford. (n.d.). *UFLDL Tutorial*. Retrieved from http://ufldl.stanford.edu: http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial