UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA

ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

" PERCEPTRÓN MONOCAPA"

Asignatura: MINERÍA DE DATOS

Docente : Prof**.** CARLOS FERNANDO MONTOYA CUBAS

Estudiante:

- CHOQUE SARMIENTO LEIDY DIANA
- ESCOBEDO MESCCO ANGIE
- ROJAS CAHUANA ETSON RONALDAO
- TTITO QUILCA CESAR RODRIGO
- TTITO SAYA ALEXANDER

Cusco - Perú

2022

RESUMEN

En los últimos años, ha habido un resurgimiento en el campo de la Inteligencia Artificial en donde replicamos el comportamiento de nuestro sistema de neuronas en distintos modelos matemáticos, conocidos todos, en su conjunto, como Red de Neuronas Artificiales(RNA). Dentro de los muchos tipos de esquemas que se han desarrollado en este ámbito podemos manifestar la evolución y el funcionamiento del modelo Perceptrón monocapa esta red es muy sencillo porque simplemente lee los valores de entrada, suma todos las entradas de acuerdo a unos pesos y el resultado lo introduce en una función de activación que genera el resultado final.

El entrenamiento del perceptrón no es más que determinar los pesos sinápticos y el umbral que mejor hagan que la entrada se ajuste a la salida. Para la determinación de estas variables, se sigue un proceso adaptativo. El proceso comienza con valores aleatorios y se van modificando estos valores según la diferencia entre los valores deseados y los calculados por la red. En resumen, el perceptrón aprende de manera iterativa.

Keywords: RNA, Perceptrón monocapa, sinápticos e iterativa

INTRODUCCIÓN

El aprendizaje automático es uno de los campos más candentes y emocionantes de la era moderna de la tecnología. Gracias al aprendizaje automático, disfrutamos de sólidos filtros de correo no deseado, reconocimiento de texto y voz conveniente, motores de búsqueda web confiables, jugadores de ajedrez desafiantes y, con suerte, pronto, automóviles autónomos seguros y eficientes.

Sin duda, el aprendizaje automático se ha convertido en un campo grande y popular y, a veces, puede ser un desafío ver el bosque (aleatorio) para los árboles (de decisión). Por lo tanto, pensé que podría valer la pena explorar diferentes algoritmos de aprendizaje automático con más detalle no solo discutiendo la teoría sino también implementando paso a paso (Raschka, 2015).

Uno de los primeros algoritmos de aprendizaje automático profundo es el perceptrón; que es un bloque de construcción básico de redes neuronales. Este está constituido por conjunto de sensores de entrada que reciben los patrones de entrada a reconocer o clasificar y una neurona de salida que se ocupa de clasificar a los patrones de entrada en dos clases, según que la salida de la misma es binaria. Sin embargo, este modelo tiene muchas limitaciones, como por ejemplo, no es capaz de aprender la función lógica XOR. (Guillen, 2015).

En este trabajo lo que nosotros mostramos es el perceptrón con utilización de la librería pyspark esto será para ver si el resultado de este trabajo tiene transferencia para producir un etiquetado.

MARCO TEÓRICO

A) Apache Spark

Apache Spark es un motor de computación en clúster de propósito general que es muy rápido y confiable. Este sistema proporciona interfaces de programación de aplicaciones en varios lenguajes de programación como Java, Python, Scala. Spark es un sistema de computación en clúster de Apache con estado de incubadora, esta herramienta está especializada en hacer que el análisis de datos sea más rápido, es bastante rápido tanto para ejecutar programas como para escribir datos. Spark es compatible con la computación en memoria, lo que le permite consultar datos mucho más rápido en comparación con los motores basados en disco como Hadoop, y también ofrece un modelo de ejecución general que puede optimizar gráficos de operadores arbitrarios.[1]

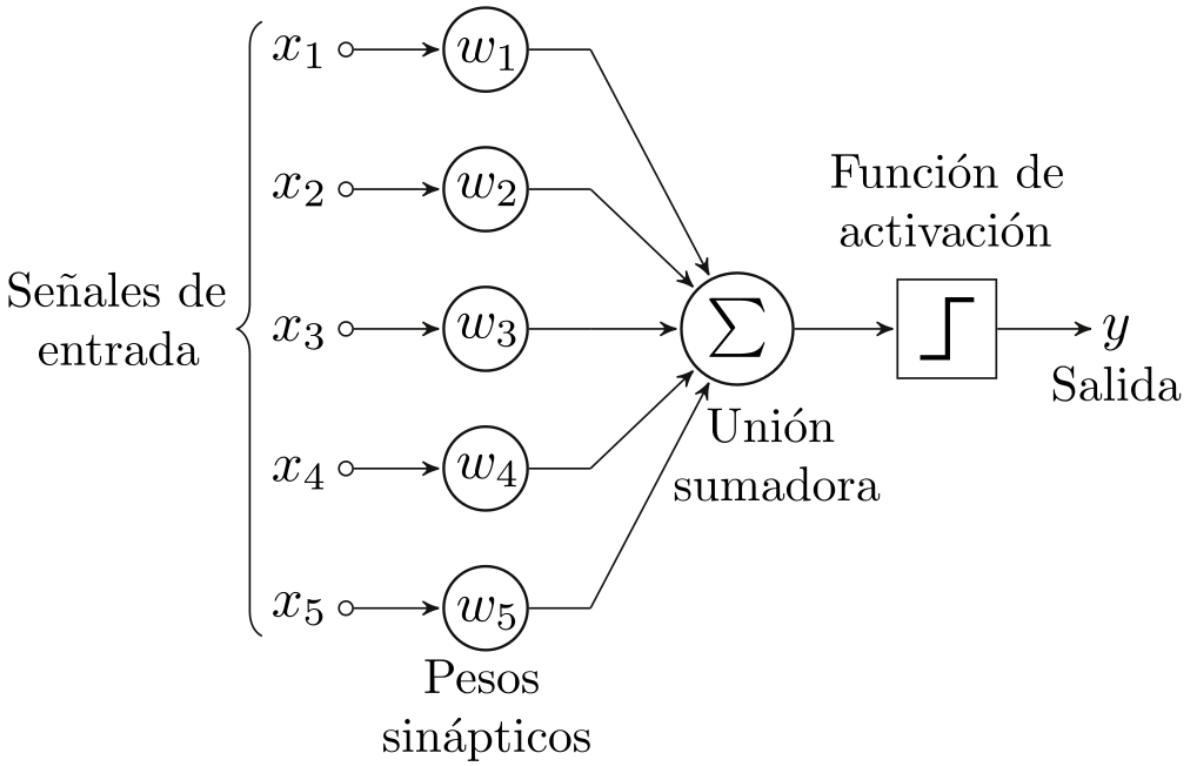
B) PySpark parallelize():

Es una función en SparkContext y se usa para crear un RDD a partir de una colección de listas. En este artículo, explicaré el uso de paralelizar para crear un RDD y cómo crear un RDD vacío con el ejemplo de PySpark.[3]

C) Perceptrón monocapa

Este modelo neuronal fue introducido por Rosenblatt a finales de los años cincuenta. La estructura del perceptrón se inspira en las primeras etapas de procesamiento de los sistemas sensoriales de los animales (por ejemplo, el de visión), en los cuales la información va atravesando sucesivas capas de neuronas, que realizan un procesamiento progresivamente de más alto nivel.[2]

El perceptrón simple es un modelo neuronal unidireccional, compuesto por dos capas de neuronas, una de entrada y otra de salida. La operación de una red de este tipo, con ‘X’ neuronas de entrada y ‘Y’ neuronas de salida, se puede expresar de la siguiente forma:



D) Algoritmos de entrenamiento

1. Inicializar los pesos (W) y el umbral (θ) en valores pequeños aleatorios

2. Leer un par de entrenamientos $X = \{X1, X2, X3, X4, X5\}$ y las salida deseadas $D = \{d1\}$

3. Calcular el producto punto

$$\Sigma XW = x1 * w1 + x2 * w2 + \ldots + xn * wn - \theta$$

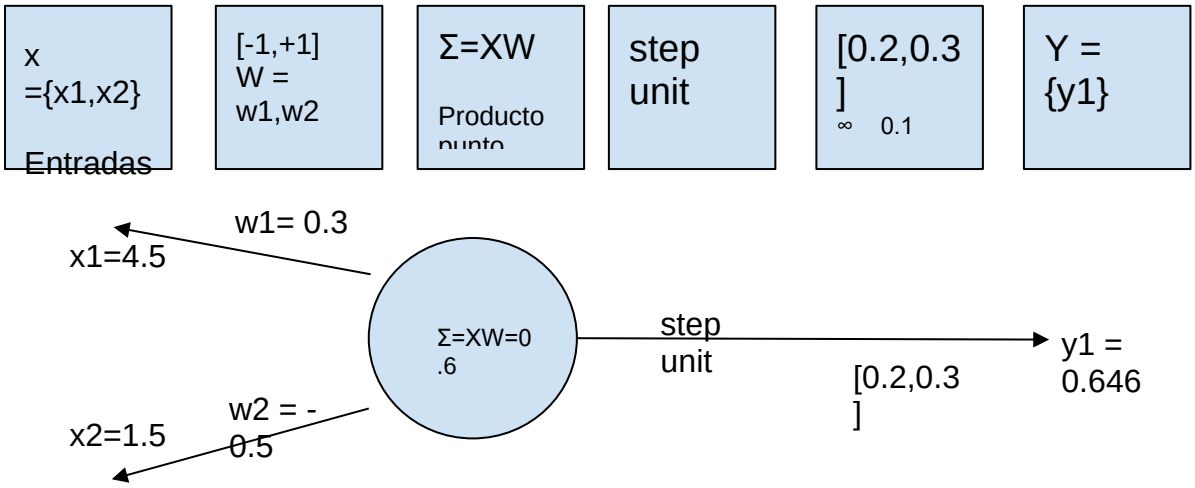
4. Aplicar la función de activación 'escalón unitario'

5. Modificar los pesos

$W_k = W_k + Coef.Aprendizaje * D - Y * x_k$, tupla → x_k

6. Regresar al paso 2 y repetir el proceso hasta tener un valor aceptable de error

Ahora Mostraremos un ejemplo de perceptrón simple:



CONSTRUCCIÓN DEL ALGORITMO

1. instalación del py spark en google colab

```
[ ] !pip install pyspark==3.0.1 py4j==0.10.9

Collecting pyspark==3.0.1
  Downloading pyspark-3.0.1.tar.gz (204.2 MB)
     |████████████████████████████████| 204.2 MB 34 kB/s
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 35.5 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.0.1-py2.py3-none-any.whl size=204612243 sha256=7c13c027450f8c9f13d7928ab590483c1ce242535ad77f4f12a25b620d155037
  Stored in directory: /root/.cache/pip/wheels/5e/34/fa/b37b5cef503fc5148b478b2495043ba61b079120b7ff379f9b
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.0.1
```

2. Importacion de librerias necesarias y creacion del SparkContext

```
import IPython
from pyspark import SparkConf
from pyspark.context import SparkContext
from pyspark.sql import SparkSession
import random
import numpy as np
import math
from pyspark.ml.linalg import DenseVector
from math import e
from pyspark.ml.feature import StringIndexer, IndexToString
from pyspark.ml.feature import VectorAssembler, VectorIndexer
```

```
[ ] sc = SparkContext.getOrCreate(SparkConf().setMaster("local[*]"))
spark = SparkSession.builder.master("local[*]").getOrCreate()

print("pyspark version:" + str(sc.version))
print("Ipython version:" + str(IPython.__version__))
```

```
pyspark version:3.0.1
Ipython version:5.5.0
```

3. Leemos el data set

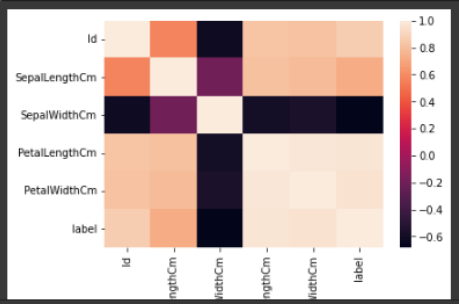
```
Iris_data = spark.read.csv("Iris_2_clases.csv", header=True, inferSchema=True)

Iris_data.show()
```

```
+---+-------------+------------+-------------+------------+-----------+
| Id|SepalLengthCm|SepalWidthCm|PetalLengthCm|PetalWidthCm|    Species|
+---+-------------+------------+-------------+------------+-----------+
|  1|          5.1|         3.5|          1.4|         0.2|Iris-setosa|
|  2|          4.9|         3.0|          1.4|         0.2|Iris-setosa|
|  3|          4.7|         3.2|          1.3|         0.2|Iris-setosa|
|  4|          4.6|         3.1|          1.5|         0.2|Iris-setosa|
|  5|          5.0|         3.6|          1.4|         0.2|Iris-setosa|
|  6|          5.4|         3.9|          1.7|         0.4|Iris-setosa|
```

4. Matriz de correlación

```
numeric_features = [t[0] for t in Iris_data.dtypes if t[1] !='string']
numeric_features_df = Iris_data.select(numeric_features)
numeric_features_df.toPandas().head()
col_names = numeric_features_df.columns
features = numeric_features_df.rdd.map(lambda row: row[0:])
#Matriz de correlacion con coeficiente de pearson
#Para poder hacer comparaciones se estandariza la covarianza, generando
corr_mat = Statistics.corr(features, method="pearson")
corr_df = pd.DataFrame(corr_mat)
corr_df.index = col_names
corr_df.columns = col_names
#Mostrar mapa de calor que es una técnica de visualización de datos
sns.heatmap(corr_df);
```



5. Separar datos para entrenamiento y test

```
train_sample = 0.60
test_sample = 0.40

(train, test) = Iris_data2.randomSplit([train_sample, test_sample], 1234)
```

6. Clase perceptron

```
class perceptron:

 def __init__(self):

   self.weights = list(np.random.rand(4))

   self.kernel = 'step'
```

```
def change_weights(self, nuevos_pesos):
  self.weights=nuevos_pesos


def predecir_manual(self,vector):
  RDD = sc.parallelize([vector,self.weights])
  dot = RDD.reduce(lambda x,y: np.array(x).dot(np.array(y)))
  return self.predict(dot,self.kernel)


def progress_bar(current_value, total):
    increments = 50
    percentual = ((current_value/ total) * 100)
    i = int(percentual // (100 / increments ))
    text = "\r[{0: <{1}}] {2}%".format('=' * i, increments, percentual)
    print(text, end="\n" if percentual == 100 else "")


def new_weights(self,tupla,learning_rate, error):
  for n in range(0, len(self.weights)):
    self.weights[n] = round(self.weights[n],7)+learning_rate*error*tupla[1][n]


def potencia(numero, exponente):
    contador = 1  # Simple ayudante del ciclo
    elevado = 1  # Aquí almacenamos el resultado de ir multiplicando el número
    # Hacer un ciclo desde 1 hasta el exponente, para multiplicarlo ese número de veces
    while contador <= exponente:
```

```
        elevado = elevado * numero
        contador = contador + 1
    return elevado

def predict(self,result_dot):
  return {'linear' : result_dot,
          'step'   : 0 if result_dot < 0 else 1,
          'sigmoid': 0 if 1/(1 + self.potencia(e,-(result_dot))) < 0 else 1}[self.kernel]

def evaluate_training(self,data_train_test_rdd):
  VP=0
  for elemento in range(1,data_train_test_rdd.count()+1):
      rdd_iterator = sc.parallelize(data_train_test_rdd.take(elemento)[elemento-1])
      tupla_actual = rdd_iterator.collect()
      #Realizar producto punto a la tupla actual y los pesos
      RDD1 = sc.parallelize([tupla_actual[1],self.weights])
      #RDD1_resultado = RDD1.reduce(lambda x,y: x[0]*y[0] + x[1]*y[1] + x[2]*y[2] +
x[3]*y[3])
      RDD1_resultado = RDD1.reduce(lambda x,y: np.array(x).dot(np.array(y)))
      #Obtener el resultado de los pesos con la funcion de activacion
      Y=self.predict(RDD1_resultado,self.kernel)
      if (tupla_actual[0] == Y):
        VP += 1
  print('Acertadas correctamente: ', VP)
  print('Total de datos: ', data_train_test_rdd.count())
```

```
  print('Precision: ',VP/data_train_test_rdd.count())



def fit(self,data_rdd, epochs, learning_rate, kernel_function):

  self.weights = list(np.random.rand(len(data_rdd.take(1)[0][1])))

  print('Pesos iniciales: ', self.weights,'\n')

  self.kernel=kernel_function

  for epoca in range(0,epochs):

    print('Epoca: ',epoca+1,'\t')

    for elemento in range(1,data_rdd.count()+1):

      self.progress_bar(elemento, data_rdd.count())

      #Obtener la segunda tupla

      rdd_iterator = sc.parallelize(data_rdd.take(elemento)[elemento-1])

      tupla_actual = rdd_iterator.collect()

      #Realizar producto punto a la tupla actual y los pesos

      RDD1 = sc.parallelize([tupla_actual[1],self.weights])

      RDD1_resultado = RDD1.reduce(lambda x,y: np.array(x).dot(np.array(y)))

      #Obtener el resultado de los pesos con la funcion de activacion

      Y=self.predict(RDD1_resultado, self.kernel)

      error=tupla_actual[0]-Y

      #Calcular los nuevos pesos

      self.new_weights(self.weights, tupla_actual, learning_rate, error)

    print('Nuevos pesos hallados: ', self.weights,'\n')
```

7. Entrenamiento del perceptron con 5 epocas

```
model = perceptron()
#fit(train, pesos, 5, 0.01,'step')
#Pesos ideales - IRIS:  [0.8162567977208124, 0.4037439312991393, 0.7589376519521082, 0.39778513745243127]
model.fit(train, epochs=5, learning_rate=0.001, kernel_function='step')
```

```
Pesos iniciales:  [0.5984166427988814, 0.358179409484781, 0.13726861650817102, 0.3620252456254939]

Epoca:  1
[==================================================] 100.0%
Nuevos pesos hallados:  [0.4131166, 0.2315794, 0.0832686, 0.3529252]

Epoca:  2
[==================================================] 100.0%
Nuevos pesos hallados:  [0.2278166, 0.1049794, 0.0292686, 0.3438252]

Epoca:  3
[==================================================] 100.0%
Nuevos pesos hallados:  [0.0425166, -0.0216206, -0.0247314, 0.3347252]

Epoca:  4
[==================================================] 100.0%
Nuevos pesos hallados:  [0.0016166, -0.0503206, -0.0366314, 0.3320252]

Epoca:  5
[==================================================] 100.0%
Nuevos pesos hallados:  [0.0016166, -0.0503206, -0.0366314, 0.3320252]
```

8. Predicion de un dato desconocido.

```
[ ]  model.predecir_manual([5.1,3.5,1.4,0.2])

     0
```

CONCLUSIÓN

Vemos que el perceptrón monocapa es una neurona sola y aislada que carece de razón de ser, puesto que solo es capaz de representar funciones lineales debido a que no dispone de capas ocultas como por ejemplo el perceptrón multicapa por eso su labor especializada se torna valiosa en la medida en que se asocia a otras neuronas, formando así una red.

En la actualidad, las redes neuronales se componen de numerosas capas de procesamiento para así obtener modelos de representación de los datos de entrada, a través de módulos simples no lineales de representación, para ello la forma más frecuente de aprendizaje automático, es el aprendizaje supervisado, en el cual el algoritmo requerirá una función objetivo, para ajustar así los pesos de modo tal que las salidas del algoritmo se acerquen a la meta que nos dispongamos.

TRABAJO A FUTURO

Durante el desarrollo del trabajo se utiliza el entorno spark de manera básica, como trabajo a futuro es mejorar algunos modulo para poder optimizar uso de recursos y tiempo evitando algunos ciclos for así trabajar netamente con grandes cantidades de datos.

BLIBLIOGRAFIA

[1] Shoro, A. G., & Soomro, T.R(2015). Big data analisys: Apache spark perspective. *Global Journal of Computer Science and Tecnology.*

[2] *Single-Layer Neural Networks and Gradient Descent*. (2015, 24 marzo). Dr. Sebastian Raschka. https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

[3] N.(2021),2,marzo. Post autor: NNN.

N. (2021),2,marzo). Post *autor*: NNK. Spark by {Examples}. https:/sparkbyexamples.com/pyspark/pysaprk-parallelize-create-rdd/