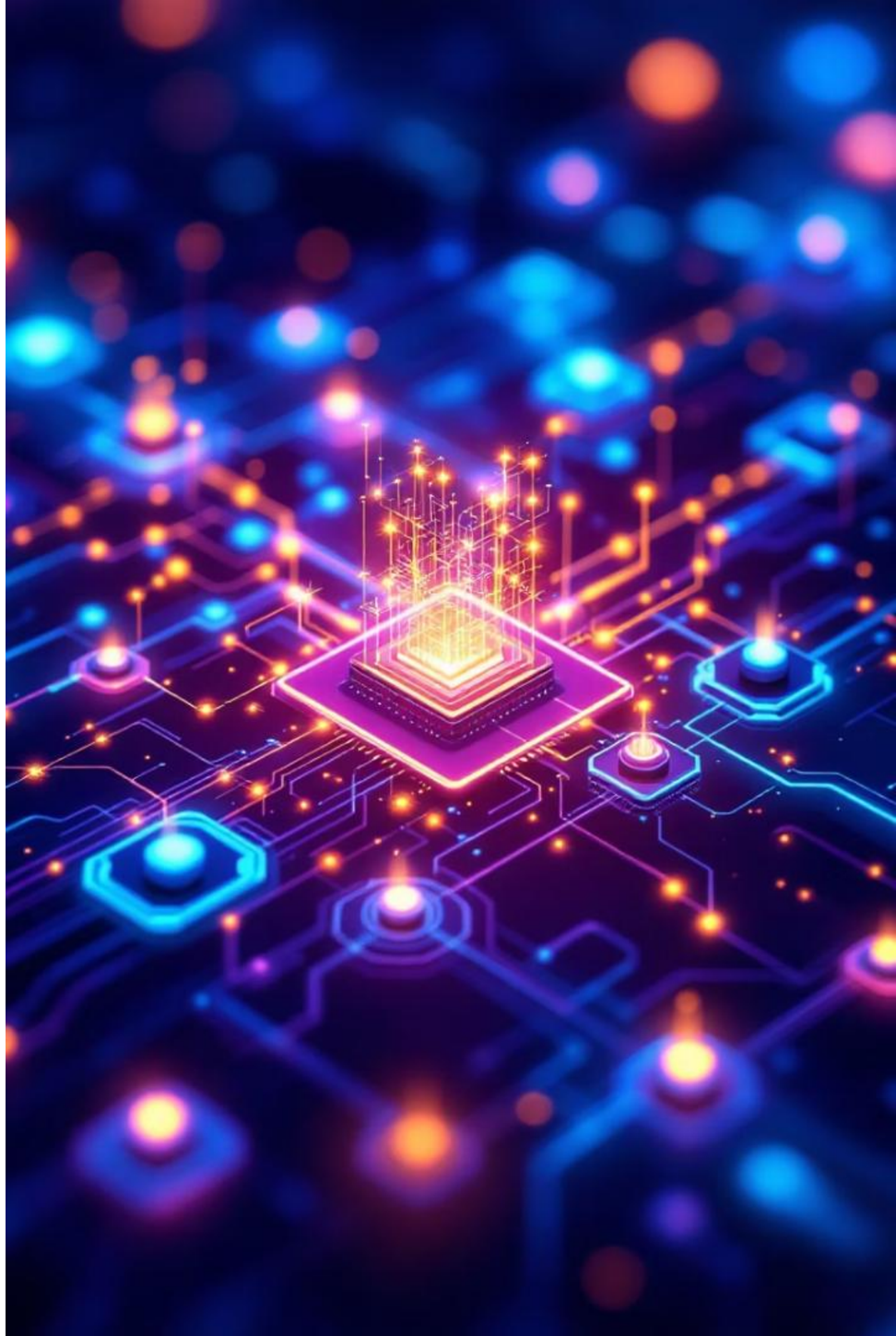
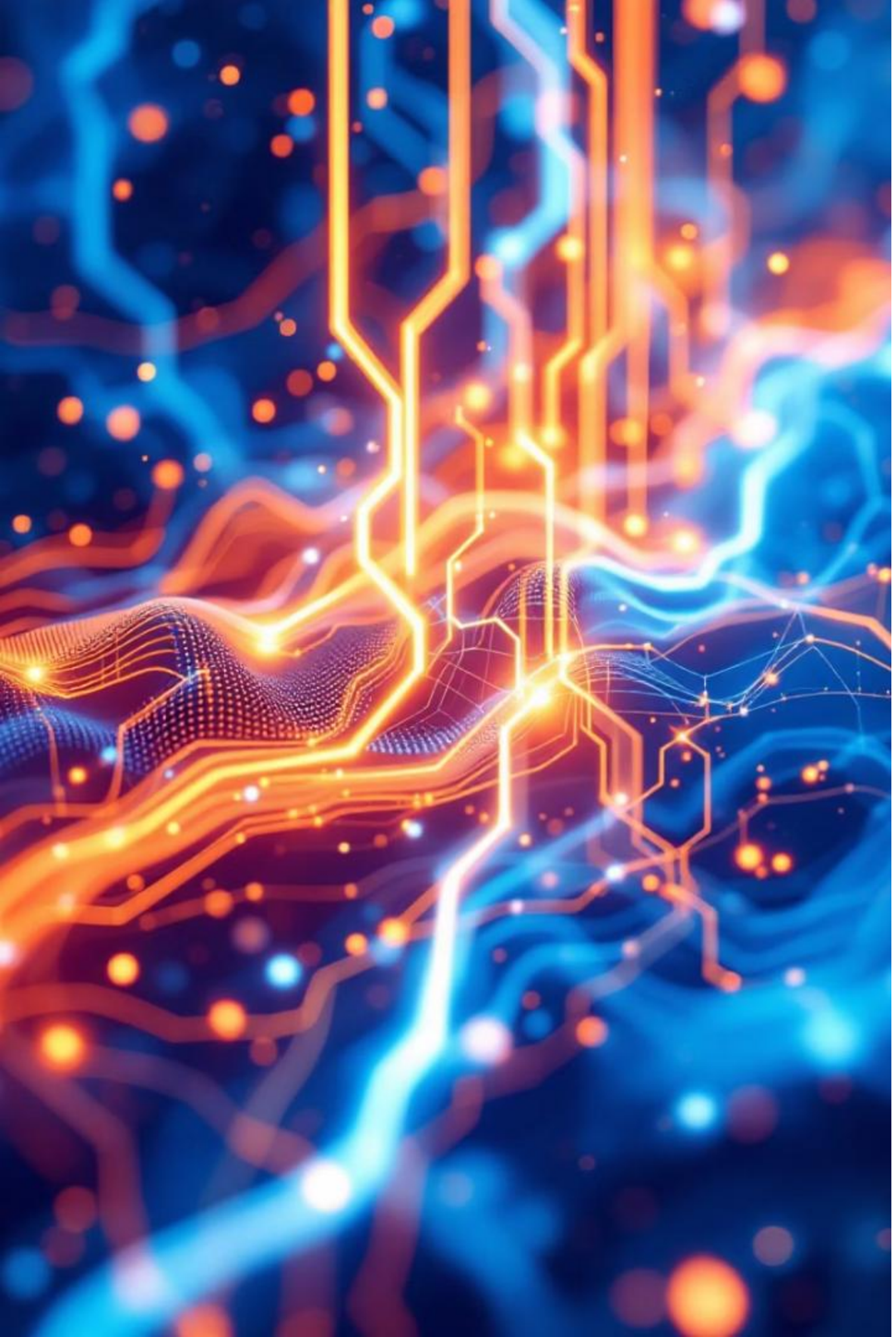
Introducción al Machine Learning Escalable

MLlib es una librería de aprendizaje automático de Apache Spark diseñada para procesar grandes volúmenes de datos. Aprovecha el procesamiento distribuido de Spark para escalar eficientemente en clústeres de computadoras, permitiendo construir y ejecutar modelos de machine learning sobre datos masivos.

Esta potente herramienta proporciona una variedad de algoritmos y utilidades que facilitan la implementación de soluciones de aprendizaje automático escalables. A lo largo de esta presentación, exploraremos sus características principales, estructuras de datos, algoritmos soportados y ejemplos prácticos de implementación.

por Kibernum Capacitación S.A.





Preguntas de Activación de Contenidos

1) ¿Qué es el procesamiento de streaming y cómo se diferencia del procesamiento por lotes?
2) ¿Qué es un DStream en Spark Streaming y qué tipo de datos representa?
3) ¿Qué es el watermarking en Spark Structured Streaming y cómo ayuda a manejar los eventos tardíos?

Características y Estructuras de Datos de MLlib

Escalabilidad

Maneja grandes volúmenes de datos distribuidos y realiza cálculos eficientemente en entornos de clústeres, ideal para tareas de big data.

Optimización y Sintonización

Incluye herramientas para selección de modelos y ajuste de hiperparámetros como cross-validation y grid search.

Facilidad de Uso

Proporciona APIs en Scala, Python, Java y R, permitiendo implementar modelos de machine learning de manera sencilla.

MLlib utiliza estructuras de datos fundamentales como RDDs y DataFrames, además de estructuras específicas como LabeledPoint, DenseVector y SparseVector. LabeledPoint representa datos etiquetados para aprendizaje supervisado, mientras que DenseVector y SparseVector representan características de manera eficiente según la densidad de los datos.

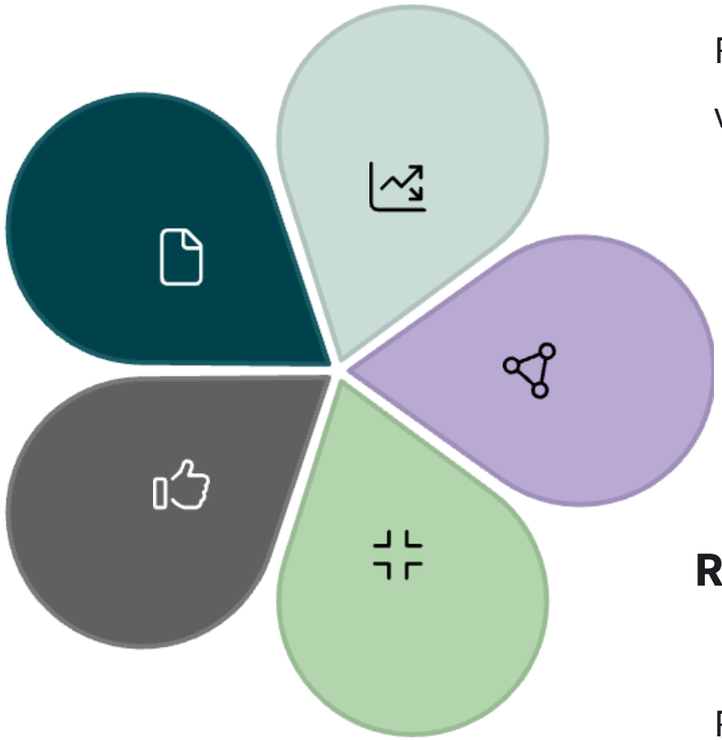
Algoritmos de Machine Learning Soportados por MLlib

Clasificación

Regresión Logística, Árboles de Decisión, Máquinas de Soporte Vectorial (SVM) y K-Vecinos más Cercanos (KNN).

Filtrado Colaborativo

ALS (Alternating Least Squares) para sistemas de recomendación basados en factorización de matrices.



Regresión

Regresión Lineal, Regresión de Lasso y Ridge para modelar relaciones entre variables y evitar sobreajuste.

Clustering

K-Means y Clustering Jerárquico para agrupar datos en clústeres según su similitud.

Reducción de Dimensionalidad

PCA (Análisis de Componentes Principales) para reducir dimensiones conservando características importantes.

Algoritmos Supervisados con MLlib

Preparación de Datos

Crear un conjunto de datos etiquetados donde cada punto tiene características y una etiqueta de salida conocida.

Entrenamiento del Modelo

Utilizar algoritmos como Regresión Logística, Árboles de Decisión, SVM o Regresión Lineal para entrenar con los datos.

Evaluación

Medir el rendimiento del modelo con métricas como precisión, recall o error cuadrático medio.

Predicción

Aplicar el modelo entrenado a nuevos datos para clasificar o predecir valores.

Los algoritmos supervisados en MLlib permiten resolver problemas de clasificación y regresión de manera eficiente en entornos distribuidos. El ejemplo de código mostrado en la presentación implementa una regresión logística para clasificación binaria, entrenando el modelo con datos etiquetados y evaluando su precisión.

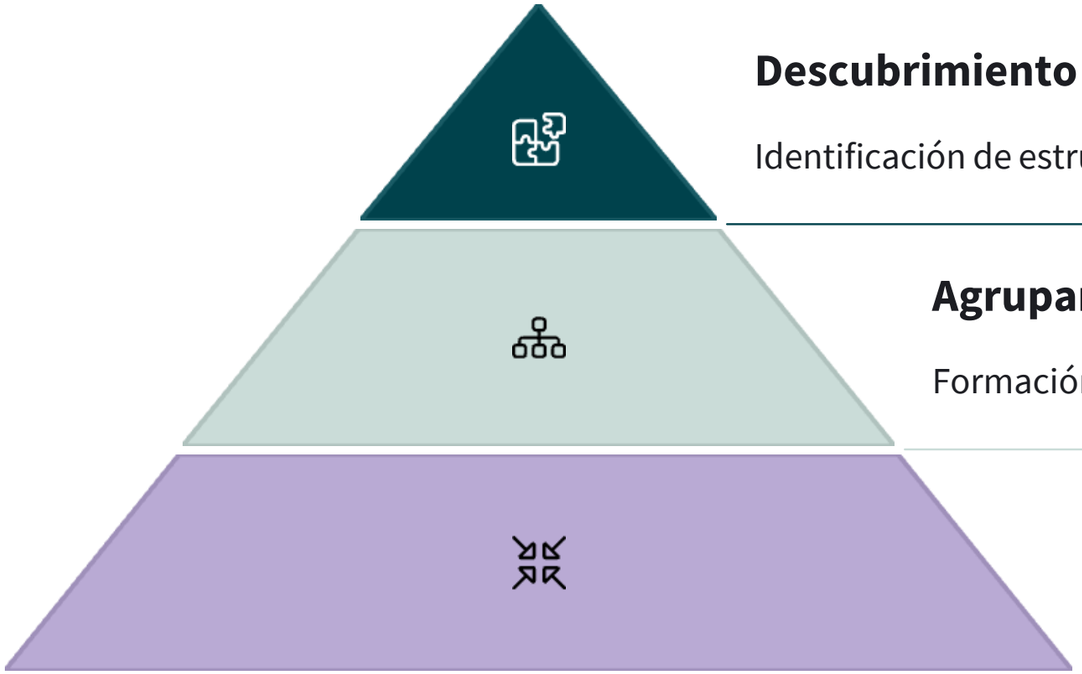
Algoritmos Supervisados con MLlib

A continuación, vamos a implementar un ejemplo simple de **regresión logística** usando **MLlib** para clasificar datos en dos categorías. Vamos a usar un conjunto de datos sintéticos para ilustrar cómo entrenar un modelo y hacer predicciones

Este ejemplo demuestra cómo usar **regresión logística** en **MLlib** para un problema de clasificación binaria. El modelo es entrenado en un conjunto de datos pequeño y luego se utiliza para hacer predicciones, evaluando su precisión con un evaluador de clasificación binaria. Esta es una forma eficiente de implementar y probar modelos de aprendizaje supervisado en un entorno distribuido utilizando Apache Spark.

```python
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import col
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Iniciar sesión de Spark
spark = SparkSession.builder.appName("LogisticRegressionExample").getOrCreate()

# Crear un DataFrame de ejemplo (con dos características y una etiqueta)
data = [(1.0, 2.0, 0.0), (2.0, 3.0, 1.0), (3.0, 4.0, 0.0), (4.0, 5.0, 1.0), (5.0, 6.0, 0.0)]
columns = ["feature1", "feature2", "label"]

# Crear el DataFrame
df = spark.createDataFrame(data, columns)

# Mostrar el DataFrame
df.show()

# Usar VectorAssembler para combinar las características en un solo vector
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
assembled_data = assembler.transform(df)

# Mostrar el DataFrame con la columna de características combinadas
assembled_data.show()

# Dividir los datos en conjunto de entrenamiento y prueba (80% entrenamiento, 20% prueba)
train_data, test_data = assembled_data.randomSplit([0.8, 0.2], seed=1234)

# Crear un modelo de regresión logística
lr = LogisticRegression(featuresCol="features", labelCol="label")

# Entrenar el modelo usando el conjunto de entrenamiento
model = lr.fit(train_data)

# Hacer predicciones usando el conjunto de prueba
predictions = model.transform(test_data)

# Mostrar las predicciones
predictions.select("features", "label", "prediction").show()

# Evaluar el modelo utilizando un evaluador de clasificación binaria
evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction")
accuracy = evaluator.evaluate(predictions)

print(f"Precisión del modelo: {accuracy:.2f}")

# Detener la sesión de Spark
spark.stop()
```

Algoritmos No Supervisados con MLlib



Descubrimiento de patrones

Identificación de estructuras ocultas en los datos

Agrupamiento de datos similares

Formación de clústeres basados en similitud

Reducción de complejidad

Simplificación de datos manteniendo información clave

Los algoritmos no supervisados en MLlib, como K-Means y PCA, no requieren datos etiquetados y buscan encontrar patrones o estructuras subyacentes en los datos. El ejemplo de implementación de K-Means mostrado en la presentación ilustra cómo realizar clustering para agrupar puntos de datos similares, visualizar los resultados y evaluar la calidad del modelo.

Estas técnicas son ampliamente utilizadas en segmentación de clientes, análisis de patrones y detección de anomalías, permitiendo descubrir información valiosa incluso cuando no se dispone de etiquetas previas.

Algoritmos No Supervisados con MLlib

En este ejemplo, implementaremos el algoritmo **K-Means** para el **clustering** de datos.

Este ejemplo ilustra cómo usar el algoritmo de K-Means de MLlib para realizar clustering en un conjunto de datos. Usamos K-Means para agrupar puntos de datos en clústeres, visualizamos los resultados y evaluamos la calidad del modelo. El uso de MLlib permite realizar estos procedimientos de forma eficiente, incluso con grandes volúmenes de datos distribuidos.
Este tipo de análisis es comúnmente utilizado en tareas de **segmentación de clientes**, **análisis de patrones** y **detección de anomalías**.

```python
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import ClusteringEvaluator
import numpy as np
import matplotlib.pyplot as plt
# Iniciar sesión de Spark
spark = SparkSession.builder.appName("KMeansExample").getOrCreate()

# Crear un conjunto de datos sintético con dos características
data = [(1.0, 2.0), (1.5, 1.8), (5.0, 8.0), (8.0, 8.0), (1.0, 0.6), (9.0, 11.0),
        (8.0, 3.0), (7.0, 4.0), (3.0, 3.0), (6.0, 7.0)]
columns = ["feature1", "feature2"]

# Crear el DataFrame
df = spark.createDataFrame(data, columns)
# Mostrar el DataFrame
df.show()
# Usar VectorAssembler para combinar las características en un solo vector
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
assembled_data = assembler.transform(df)

# Mostrar el DataFrame con la columna de características combinadas
assembled_data.show()
# Crear el modelo de K-Means, especificando el número de clústeres
kmeans = KMeans().setK(2).setSeed(1)
# Entrenar el modelo de K-Means
model = kmeans.fit(assembled_data)
# Hacer las predicciones (es decir, asignar cada punto de datos a un clúster)
predictions = model.transform(assembled_data)

# Mostrar las predicciones (los clústeres asignados a cada punto de datos)
predictions.show()

# Obtener los centroides de los clústeres
centers = model.clusterCenters()
print("Centroides de los clústeres: ")
for center in centers:
    print(center)

# Visualización de los resultados usando matplotlib (para 2D)
pred_data = predictions.select("feature1", "feature2", "prediction").toPandas()

plt.scatter(pred_data['feature1'], pred_data['feature2'], c=pred_data['prediction'], cmap='viridis')
plt.scatter([center[0] for center in centers], [center[1] for center in centers], s=200, c='red',
marker='x')  # Centroides
plt.title("Clustering K-Means")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Evaluación del modelo usando ClusteringEvaluator (índice de silueta)
evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(predictions)
print(f"Índice de Silueta del Clustering: {silhouette:.2f}")
# Detener la sesión de Spark
spark.stop()
```

Actividad Práctica Guiada

OBJETIVO: En esta actividad, desarrollaremos 2 algoritmos de aprendizaje supervisado y 2 de aprendizaje no supervisado. Para cada algoritmo, elaboraremos un caso de uso práctico en el que sea aplicable y explicaremos por qué el algoritmo es adecuado para ese caso.

ALGORITMOS SUPERVISADOS

Algoritmo Regresión Lineal:
•**Caso**: Predicción del precio de una casa basada en sus características.
•**Razón**: Es un modelo adecuado para predecir variables continuas, como el precio de una casa, basándose en una relación lineal entre las características.

Árbol de Decisión:
•**Caso**: Clasificación de usuarios en compradores y no compradores en un sitio web.
•**Razón**: Ideal para clasificación binaria, proporcionando una visualización clara y fácil de entender de las decisiones.

ALGORITMOS NO SUPERVISADOS

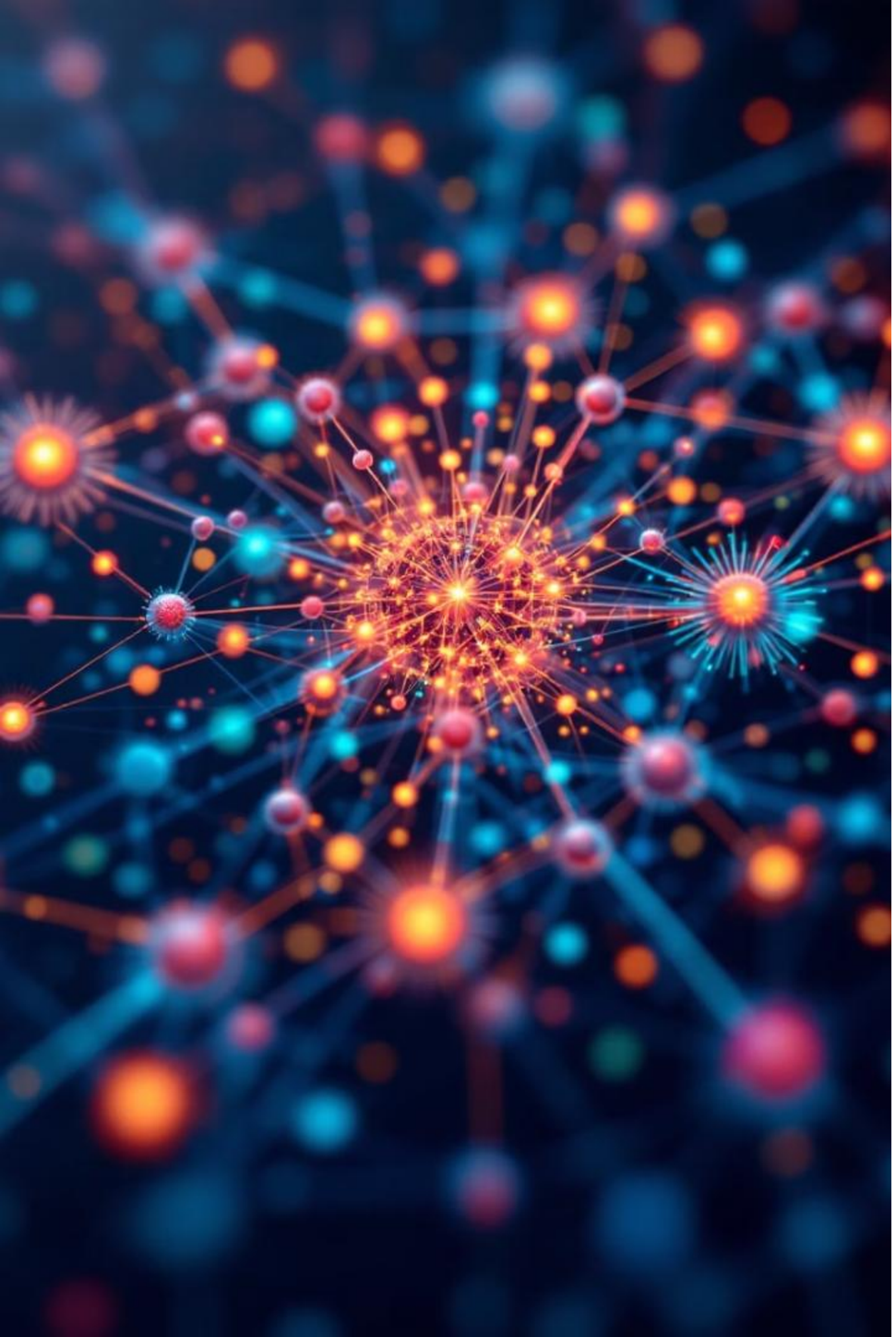
Algoritmo K-Means (Clustering):
•**Caso**: Segmentación de clientes en un supermercado para personalizar ofertas.
•**Razón**: Útil para identificar grupos naturales en los datos y ayudar a personalizar estrategias de marketing.

Algoritmo PCA (Análisis de Componentes Principales):
•**Caso**: Reducción de dimensionalidad para reconocimiento facial.
•**Razón**: Permite reducir la cantidad de características de un conjunto de datos de imágenes, facilitando el procesamiento y mejora de la eficiencia del sistema.

Enlace a Material Complementario

Uso de Apache Spark MLlib para compilar una aplicación de aprendizaje automático y analizar un conjunto de datos

https://learn.microsoft.com/es-es/azure/hdinsight/spark/apache-spark-machine-learning-mllib-ipython



Preguntas de Reflexión Final

1) ¿Cómo crees que el uso de MLlib de Spark mejora el rendimiento de los algoritmos de machine learning en comparación con un procesamiento tradicional?
2) En el caso de usar algoritmos de clustering, como **K-Means**, ¿por qué es importante elegir correctamente el número de clústeres y cómo afectaría un número incorrecto de clústeres al resultado final?
3) ¿Qué algoritmo seleccionarías para predecir si un cliente realizará una compra en línea y por qué?