Conceptos y Aplicaciones de Big Data

MLlib y GraphX

Prof. Waldo Hasperué
(whasperue@lidi.info.unlp.edu.ar)

Temario

- MLlib
- GraphX

Inteligencia de datos

- La inteligencia de datos es el proceso de analizar datos complejos y presentar información de tal manera que ayude a los usuarios a tomar decisiones de negocio basados en la información analizada y en tiempo real.

- La inteligencia de negocios es la recolección de resultados arrojados por muchos programas y sistemas que analizan flujos de datos de gran tamaño y se utiliza para generar información esencial que ayude al proceso de toma de decisiones.

Inteligencia de datos

La inteligencia de datos y de negocios puede agregar valor a cualquier proceso de negocios.

- Marketing digital
- Aumento de las tasas de membresía
- Maximización de la eficiencia operativa
- Vista unificada de operaciones
- Identificación a estudiantes en riesgo

https://www.tableau.com/es-es/learn/articles/business-intelligence-examples

Machine learning

Un sistema inteligente es aquel sistema capaz de resolver problemas complejos y multidisciplinares de una forma automática dando soporte a las decisiones de un experto.

- Algoritmos simbolistas. Razonamiento inductivo
- Redes neuronales artificiales
- Algoritmos genéticos y evolutivos
- Probabilísticos. Teorema de Bayes.
- Algoritmos de "similitudes". K-NN, SVM

Sistemas inteligentes en Big Data

- Los algoritmos que implementan los sistemas inteligentes son algoritmos iterativos, por lo tanto tienen que dar varias "pasadas" a los datos para llevar a cabo su tarea.

- Se los conocen como algoritmos de aprendizaje de máquina (machine learning).

- Deben estar optimizados para un óptimo rendimiento.

MLlib

- MLlib es la librería de algoritmos de machine learning para Spark.

- Los algoritmos están diseñados e implementados para ejecutarse de manera eficiente en un ambiente distribuido

MLlib - Algoritmos

- Logistic regression
- Naive Bayes
- Generalized linear regression
- Survival regression
- Decision trees
- Random forests
- Gradient-boosted trees
- Alternating least squares (ALS)
- K-means
- Gaussian mixtures
- Latent Dirichlet allocation (LDA)
- Frequent itemsets
- Association rules
- Sequential pattern mining

MLlib – Ejemplo de uso

```
from pyspark.ml.regression import LinearRegression

# Carga de los datos
datos = sc.textFile("datos.txt")

lr = LinearRegression(maxIter=10, regParam=0.01,
                      elasticNetParam=0.01)

# Entrenamiento del modelo
lrModel = lr.fit(datos)

# Resultados
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))
```

MLlib – Ejemplo de uso

```
from pyspark.ml.classification import
                                DecisionTreeClassifier

# Carga de los datos
datos = sc.textFile("datos.txt")

dt = DecisionTreeClassifier(labelCol="indexedLabel",
                        featuresCol="indexedFeatures")

# Entrenamiento del modelo
modelo = dt.fit(datos)

##############################################
# Predicciones
nuevos = sc.textFile("nuevos.txt")
predicciones = modelo.transform(nuevos)
```

MLlib – Más ejemplos

Decenas de ejemplos listos para copiar, pegar y ejecutar en:

- Python
- Java
- Scala

https://spark.apache.org/docs/3.0.0-preview/ml-guide.html

Transformer y estimators

- Estimator: es una abstracción del concepto de un algoritmo de aprendizaje. Recibe como entrada un DataFrame y produce como salida un modelo de los datos.

- Transformer: es una abstracción que recibe como entrada un DataFrame y un modelo (estimator) y devuelve como salida otro DataFrame, agregando una o más columnas. Ej: categoría de clasificación o resultado de una regresión o una predicción.

Estimator

```
class MyEstimator(Estimator):

    def fit(self, dataframe):
        # Análisis del dataframe
            . . .
        # Creación del modelo
            . . .
        model = MyModel()
        model.setParams(modelParams)
        return model
```

Transformer

```
class MyModel(Model):
    __params = null
    def setParams(self, modelParams):
        self.__params = modelParams

    def transform(self, dataframe):
        # Uso del modelo para agregar
        # la respuesta
        return dataframe.withColumn(…)
```

Ejemplo

- Un supermercado tiene información de sus clientes. De cada cliente se tiene la categoría a la que pertenece (stándard o premium) y el total de dinero anual destinado a las compras en el supermercado.
- A principio de año, el supermercado desea reasignar las categorías a sus clientes según lo que gastó cada uno el último año.

Ejemplo

Para ello, ordena a sus clientes por dinero gastado y establece el nuevo umbral de corte como el punto medio entre el cliente estándar que más gastó y el cliente premium que menos gastó.

Categoría	Monto
Premium	956961
Premium	938936
Premium	845657
Estándar	830698
Estándar	790676
Premium	550083
Estándar	469892
Estándar	329549
Premium	133364
Estándar	59049
Estándar	35280

$$\frac{830698 + 133364}{2} = \mathbf{482031}$$

Ejemplo

Todos los clientes con dinero gastado mayor al nuevo punto de corte se convierten en premium, el resto pasa a ser estándar.

Categoría	Monto
Premium	956961
Premium	938936
Premium	845657
Estándar	830698
Estándar	790676
Premium	550083
Estándar	469892
Estándar	329549
Premium	133364
Estándar	59049
Estándar	35280

$$\frac{830698 + 133364}{2} = \mathbf{482031}$$

Ejemplo - Estimator

```
from pyspark.ml.pipeline import Estimator
class NuevaCategoriaEstimator(Estimator):
  def fit(self, dataframe):
    # Análisis del dataframe. Creación del modelo
    premium = dataframe.filter(dataframe.categoria == "Premium")
    estandar = dataframe.filter(dataframe.categoria == "Estándar")
    minimo = premium.select(F.min(premium.monto))
                .withColumnRenamed('min(monto)','min').first()
    maximo = estandar.select(F.max(premium.monto))
                .withColumnRenamed('max(monto)','max').first()
    corte = (minimo.min + maximo.max) / 2
    model = MyModel()
    model.setCorte(corte)
    return model
```

Ejemplo - Transform

```
from pyspark.ml.pipeline import Model
from pyspark.sql.functions import lit
def _asignarCategoria(corte, monto):
     if(monto > corte):
          return "Premium"
     else:
          return "Estándar"
asignarCategoria = udf(_asignarCategoria, StringType())
class MyModel(Model):
     __corte = 0
     def setCorte(self, c):
          self.__corte = c
     def transform(self, dataframe):
          # Uso del modelo para agregar la respuesta
          return dataframe.withColumn('Nueva categoría',
                         asignarCategoria(lit(self.__corte),
                                                dataframe.monto))
```

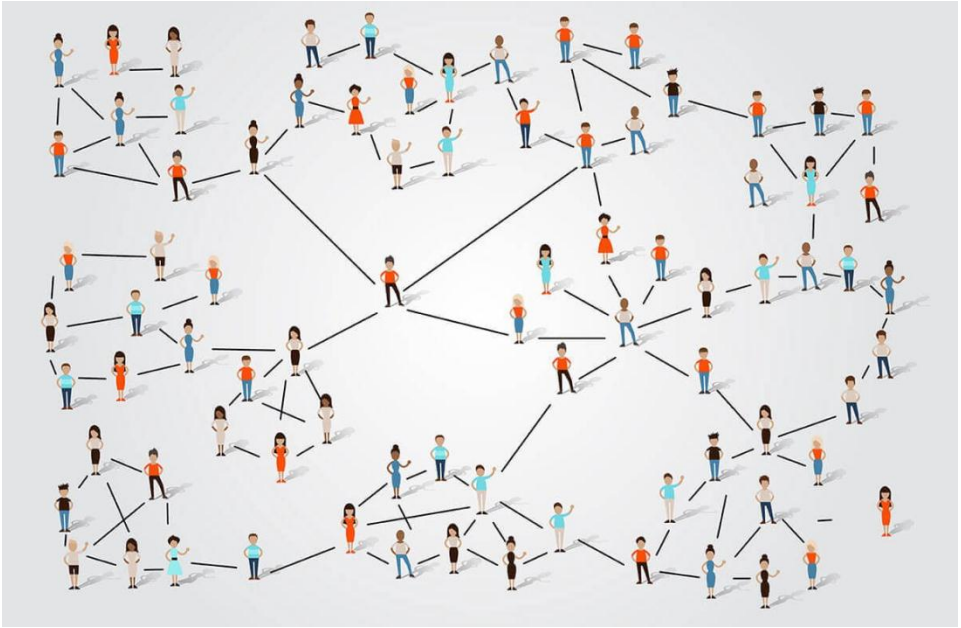
Ejemplo – Uso

```
clientes = sc.textFile("Mis datos")

estimator = NuevaCategoriaEstimator()

# Entrenamiento del modelo
modelo = estimator.fit(clientes)

##########################################
# Asignación de nuevas categorías
nuevos = sc.textFile("Nuevos")

nuevasCategorias = modelo.transform(nuevos)
nuevasCategorias.show()
```

GraphX

- GraphX es la librería de algoritmos para problemas con grafos de Spark.

- Los algoritmos están diseñados e implementados para ejecutarse de manera eficiente en un ambiente distribuido



GraphX - Algoritmos

- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count

GraphX - Ejemplos

```
from graphframes import GraphFrame

# Carga del grafo desde archivo
vertices = sc.textFile("vertices.txt")
edges = sc.textFile("edges.txt")

g = GraphFrame(vertices, edges)
print(g.inDegrees, g.outDegrees)

res = g.bfs( from, to )

pr = g.pageRank(resetProbability=0.15, tol=0.01)

cmc = g.shortestPaths(landmarks = ["a", "d"])
```

Más ejemplos de GraphX

https://spark.apache.org/docs/latest/graphx-programming-guide.html

https://github.com/graphframes/graphframes