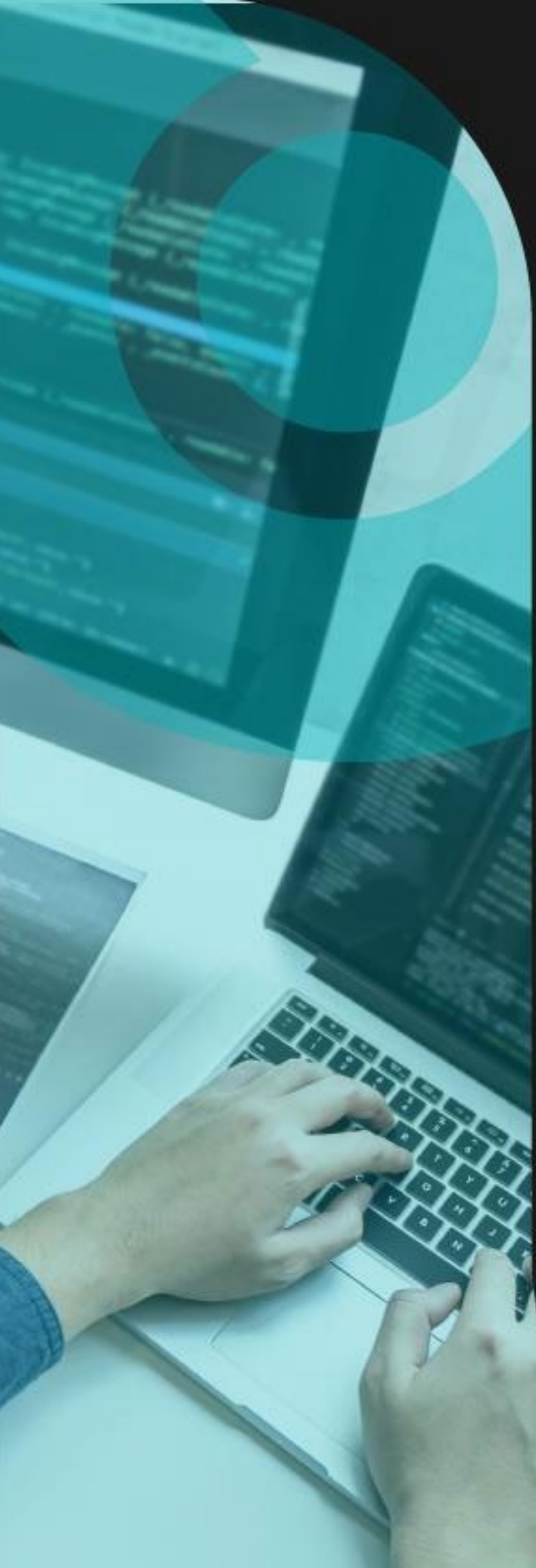


AWAKELAB

BASECAMP

Ciencia de Datos

Módulo: Fundamentos del Big Data

Aprendizaje Esperado

---

4. Elabora un modelo de aprendizaje de máquina utilizando MLlib para resolver un problema de grandes volúmenes de datos

---

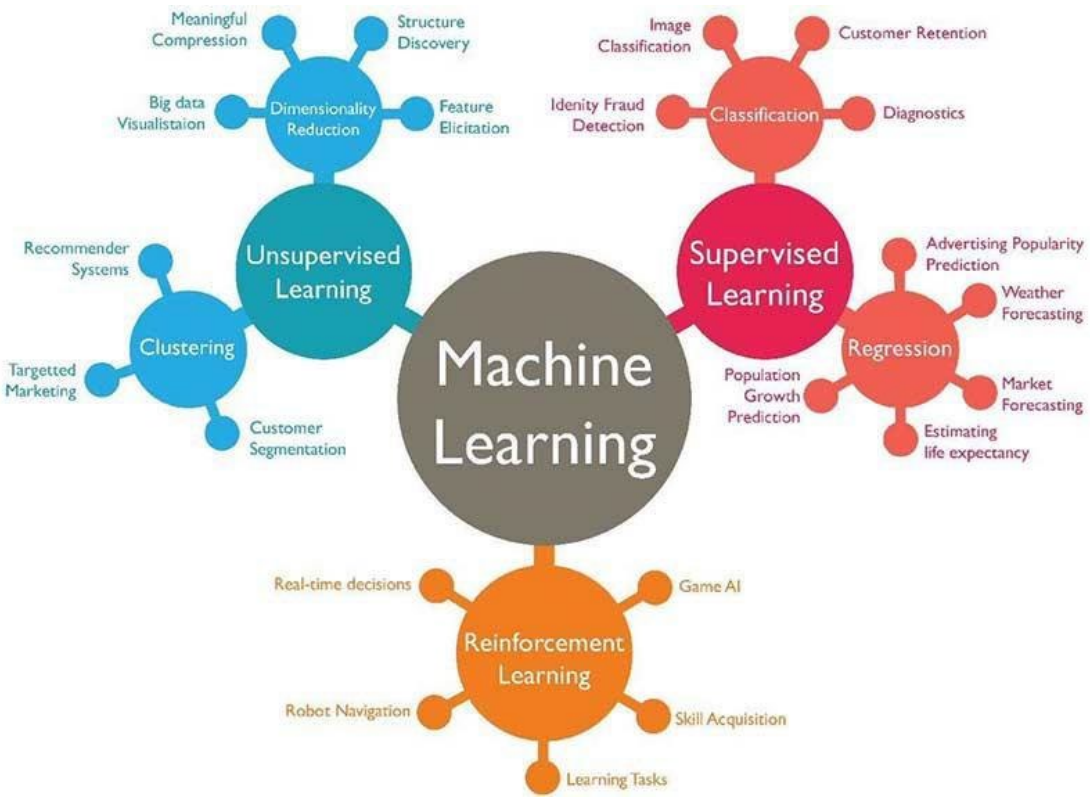
Introducción a Machine Learning

Hemos visto Machine Learning como una palabra de moda en los últimos años, la razón de esto podría ser la gran cantidad de producción de datos por parte de las aplicaciones, el aumento del poder de cómputo en los últimos años y el desarrollo de mejores algoritmos.

El aprendizaje automático se utiliza en cualquier lugar, desde la automatización de tareas mundanas hasta la oferta de información inteligente, las industrias de todos los sectores intentan beneficiarse de él. Es posible que ya esté utilizando un dispositivo que lo utilice. Por ejemplo, un rastreador de actividad física portátil como Fitbit o un asistente doméstico inteligente como Google Home. Pero hay muchos más ejemplos de ML en uso.

- Predicción: el aprendizaje automático también se puede utilizar en los sistemas de predicción. Considerando el ejemplo del préstamo, para calcular la probabilidad de falla, el sistema necesitará clasificar los datos disponibles en grupos.
- Reconocimiento de imágenes: el aprendizaje automático también se puede utilizar para la detección de rostros en una imagen. Hay una categoría separada para cada persona en una base de datos de varias personas.

- Reconocimiento de voz: es la traducción de palabras habladas al texto. Se utiliza en búsquedas por voz y más. Las interfaces de usuario de voz incluyen marcación por voz, enrutamiento de llamadas y control de dispositivos. También se puede utilizar una entrada de datos simple y la preparación de documentos estructurados.
- Diagnósticos médicos: ML está capacitado para reconocer tejidos cancerosos.
- Industria financiera y comercio: las empresas usan ML en investigaciones de fraude y verificaciones de crédito.



Aprendizaje automático

La biblioteca de aprendizaje automático de Apache Spark (MLlib) está diseñada para brindar simplicidad, escalabilidad y fácil integración con otras herramientas. Con la escalabilidad, la compatibilidad de idiomas y la velocidad de Spark, los científicos de datos pueden centrarse en sus problemas y modelos de datos en lugar de resolver las complejidades que rodean a los datos distribuidos (como infraestructura, configuraciones, etc.).

Construida sobre Spark, MLlib es una biblioteca de aprendizaje automático escalable que consta de utilidades y algoritmos de aprendizaje comunes, que incluyen clasificación, regresión, agrupación, filtrado colaborativo, reducción de la dimensionalidad y primitivas de optimización subyacentes. Spark MLLib se integra a la perfección con otros componentes de Spark, como Spark SQL, Spark Streaming y DataFrames, y se instala en el tiempo de ejecución de Databricks.

La biblioteca se puede utilizar en Java, Scala, y Python como parte de las aplicaciones de Spark, para que puedas incluirlo en flujos de trabajo completos. MLlib permite el preprocesamiento, la manipulación, el entrenamiento de modelos y la realización de predicciones a escala sobre los datos. Incluso puede usar modelos entrenados en MLlib para hacer predicciones en transmisión estructurada. Spark proporciona una API de aprendizaje automático sofisticada para realizar una variedad de tareas de aprendizaje automático, desde la clasificación hasta la regresión, la agrupación en clústeres y el aprendizaje profundo.

Qué es MLlib

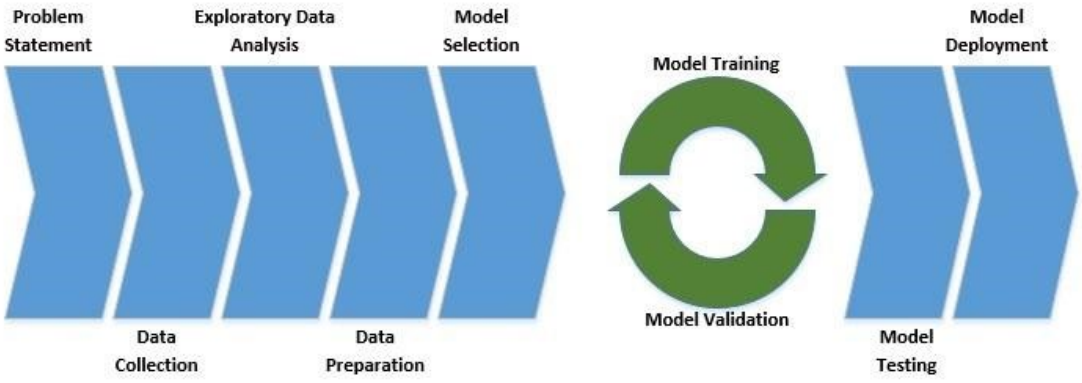
Llib es la biblioteca de aprendizaje automático (ML) de Spark. Su objetivo es hacer que el aprendizaje automático práctico sea escalable y fácil. A un alto nivel, proporciona herramientas como:

- Algoritmos de ML: algoritmos de aprendizaje comunes como clasificación, regresión, agrupamiento y filtrado colaborativo

- Caracterización: extracción de características, transformación, reducción de dimensionalidad y selección
- Pipelines: herramientas para construir, evaluar y ajustar ML Pipelines
- Persistencia: guardar y cargar algoritmos, modelos y Pipelines
- Utilidades: álgebra lineal, estadística, manejo de datos, etc.

Estructuras de dato

El aprendizaje automático es verdaderamente un área de estudio interdisciplinaria. Requiere conocimiento del dominio comercial, estadística, probabilidad, álgebra lineal y programación. Como esto claramente puede volverse abrumador, **es mejor abordarlo de manera ordenada** , lo que normalmente llamamos un flujo de trabajo de aprendizaje automático:



Aprendizaje automático con Mllib

Ahora tenemos suficiente contexto sobre el aprendizaje automático y cómo MLlib puede ayudar en este esfuerzo. Comencemos con nuestro ejemplo básico de implementación de un proyecto de aprendizaje automático con Spark MLlib.

Objetivo: **¿podemos predecir la especie de un Iris a partir de la longitud y el ancho de su sépalo y pétalo?**

Implementación supervisada

1. **Establecer las dependencias**

   Primero, tenemos que definir la siguiente dependencia en Maven para extraer las bibliotecas relevantes:

```
<dependency>

    <groupId>org.apache.spark</groupId>
    <artifactId>spark-mllib_2.11</artifactId>
    <version>2.4.3</version>
    <scope>provided</scope>
</dependency>
```

   Y necesitamos inicializar SparkContext para que funcione con las API de Spark:

```
SparkConf conf = new SparkConf()

  .setAppName("Main")
```

```
  .setMaster("local[2]");

JavaSparkContext sc = new JavaSparkContext(conf);
```

2. **Cargando los datos**

   Lo primero es lo primero, debemos descargar los datos, que están disponibles como un archivo de texto en formato CSV. Luego tenemos que cargar estos datos en Spark:

```
String dataFile = "data\\iris.data";

JavaRDD<String> data = sc.textFile(dataFile);
```

   Spark MLlib ofrece varios tipos de datos, tanto locales como distribuidos, para representar los datos de entrada y las etiquetas correspondientes. El más simple de los tipos de datos es *Vector* :

```
JavaRDD<Vector> inputData = data

  .map(line -> {

      String[] parts = line.split(",");

      double[] v = new double[parts.length - 1];
```

```
        for (int i = 0; i < parts.length - 1; i++) {
            v[i] = Double.parseDouble(parts[i]);
        }

        return Vectors.dense(v);

});
```

Tenga en cuenta que aquí solo hemos incluido las características de entrada, principalmente para realizar análisis estadísticos.
Un ejemplo de entrenamiento generalmente consta de varias características de entrada y una etiqueta, representada por la clase *LabeledPoint* :

```
Map<String, Integer> map = new HashMap<>();

map.put("Iris-setosa", 0);

map.put("Iris-versicolor", 1);

map.put("Iris-virginica", 2);



JavaRDD<LabeledPoint> labeledData = data
  .map(line -> {
```

```
        String[] parts = line.split(",");
        double[] v = new double[parts.length - 1];

        for (int i = 0; i < parts.length - 1; i++) {
            v[i] = Double.parseDouble(parts[i]);
        }

        return new
LabeledPoint(map.get(parts[parts.length - 1]),
Vectors.dense(v));
});
```

Nuestra etiqueta de salida en el conjunto de datos es textual, lo que significa la especie de Iris. Para introducir esto en un modelo de aprendizaje automático, tenemos que convertirlo en valores numéricos.

3. Análisis exploratorio de datos

El análisis exploratorio de datos implica analizar los datos disponibles. Ahora, los algoritmos de aprendizaje automático son sensibles a la calidad de los datos , por lo tanto, los datos de mayor calidad tienen mejores perspectivas de ofrecer el resultado deseado.

Los objetivos de análisis típicos incluyen la eliminación de anomalías y la detección de patrones. Esto incluso alimenta los pasos críticos de la ingeniería de funciones para llegar a funciones útiles a partir de los datos disponibles.

Nuestro conjunto de datos, en este ejemplo, es pequeño y está bien formado. Por lo tanto, no tenemos que permitirnos un montón de análisis de datos. Spark MLlib, sin embargo, está equipado con API para ofrecer una gran perspectiva. Comencemos con un análisis estadístico simple:

```
MultivariateStatisticalSummary summary =
Statistics.colStats(inputData.rdd());
       System.out.println("Summary Mean:");
System.out.println(summary.mean());
System.out.println("Summary Variance:");
System.out.println(summary.variance());
System.out.println("Summary Non-zero:");
System.out.println(summary.numNonzeros());
```

Aquí, estamos observando la media y la varianza de las características que tenemos. Esto es útil para determinar si necesitamos realizar la normalización de las funciones. Es **útil tener todas las características en una escala similar** . También estamos tomando nota de los valores distintos de cero, que pueden afectar negativamente el rendimiento del modelo.

Aquí está la salida para nuestros datos de entrada:

```
Summary Mean:

[5.843333333333332,3.0540000000000003,3.7586666666666
666,1.1986666666666668]

Summary Variance:
```

```
[0.6856935123042509,0.18800402684563744,3.11317941834
4516,0.5824143176733783]


Summary Non-zero:

[150.0,150.0,150.0,150.0]
```

Otra métrica importante para analizar es la correlación entre las características en los datos de entrada:

```
Matrix correlMatrix =
Statistics.corr(inputData.rdd(), "pearson");

System.out.println("Correlation Matrix:");

System.out.println(correlMatrix.toString());
```

Una **alta correlación entre dos características sugiere que no están agregando ningún valor incremental** y que una de ellas se puede descartar. Así es como nuestras características están correlacionadas:

```
Correlation Matrix:

1.0                  -0.10936924995064387  0.8717541573048727   0.8179536333691672

-0.10936924995064387  1.0                   -0.4205160964011671  -0.3565440896138163

0.8717541573048727    -0.4205160964011671   1.0                  0.9627570970509661

0.8179536333691672    -0.3565440896138163   0.9627570970509661   1.0
```

4. **Dividir los datos**

   Si recordamos nuestra discusión sobre el flujo de trabajo de aprendizaje automático, implica varias iteraciones de entrenamiento y validación del modelo seguidas de una prueba final.
   Para que esto suceda, tenemos que dividir nuestros datos de entrenamiento en conjuntos de entrenamiento, validación y prueba . Para simplificar las cosas, omitiremos la parte de validación. Entonces, dividamos nuestros datos en conjuntos de entrenamiento y prueba:

```
JavaRDD<LabeledPoint>[] splits =
parsedData.randomSplit(new double[] { 0.8, 0.2 },
11L);

JavaRDD<LabeledPoint> trainingData = splits[0];


JavaRDD<LabeledPoint> testData = splits[1];
```

5. **Entrenamiento modelo**

   Entonces, hemos llegado a una etapa en la que hemos analizado y preparado nuestro conjunto de datos. ¡Todo lo que queda es introducir esto en un modelo y comenzar la magia! Bueno, es más fácil decirlo que hacerlo. Necesitamos elegir un algoritmo adecuado para nuestro problema; recuerde las diferentes categorías de aprendizaje automático de las que hablamos anteriormente.

No es difícil entender que nuestro problema se encuadra dentro de la categoría supervisada . Ahora, hay bastantes algoritmos disponibles para usar en esta categoría.

El más simple de ellos es la Regresión Logística (que la palabra regresión no nos confunda, es, después de todo, un algoritmo de clasificación):

```
LogisticRegressionModel model = new
LogisticRegressionWithLBFGS()
  .setNumClasses(3)

  .run(trainingData.rdd());
```

Aquí, estamos utilizando un clasificador basado en BFGS de memoria limitada de tres clases. Los detalles de este algoritmo están más allá del alcance de este tutorial, pero este es uno de los más utilizados.

6. **Evaluación del modelo**

Recuerde que el entrenamiento del modelo involucra múltiples iteraciones, pero para simplificar, aquí solo usamos un solo paso. Ahora que hemos entrenado nuestro modelo, es hora de probar esto en el conjunto de datos de prueba:

```
JavaPairRDD<Object, Object> predictionAndLabels =
testData
```

```
  .mapToPair(p -> new
Tuple2<>(model.predict(p.features()), p.label()));

MulticlassMetrics metrics = new
MulticlassMetrics(predictionAndLabels.rdd());

double accuracy = metrics.accuracy();

System.out.println("Model Accuracy on Test Data: " +
accuracy);
```

Ahora bien, ¿cómo medimos la eficacia de un modelo? Hay **varias métricas que podemos utilizar, pero una de las más sencillas es Accuracy** . En pocas palabras, la precisión es una relación entre el número correcto de predicciones y el número total de predicciones. Esto es lo que podemos lograr en una sola ejecución de nuestro modelo:

```
Model Accuracy on Test Data: 0.9310344827586207
```

Tenga en cuenta que esto variará ligeramente de una ejecución a otra debido a la naturaleza estocástica del algoritmo.
Sin embargo, la precisión no es una métrica muy efectiva en algunos dominios de problemas. Otras **métricas más sofisticadas son Precision and Recall (F1 Score), ROC Curve y Confusion Matrix** .

7. **Guardar y cargar el modelo**

Finalmente, a menudo necesitamos guardar el modelo entrenado en el sistema de archivos y cargarlo para la predicción de los datos de producción. Esto es trivial en Spark:

```
model.save(sc, "model\\logistic-regression");

LogisticRegressionModel sameModel =
LogisticRegressionModel

  .load(sc, "model\\logistic-regression");

Vector newData = Vectors.dense(new
double[]{1,1,1,1});

double prediction = sameModel.predict(newData);

System.out.println("Model Prediction on New Data = "
+ prediction);
```

Entonces, estamos guardando el modelo en el sistema de archivos y volviéndolo a cargar. Después de cargar, el modelo se puede usar de inmediato para predecir la salida de nuevos datos. Aquí hay una predicción de muestra sobre nuevos datos aleatorios:

```
Model Prediction on New Data = 2.0
```

(Implementación tomada de: https://www.baeldung.com/spark-mlib-machine-learning).

Spark MLlib en comparación

Si bien Spark MLlib es una biblioteca bastante poderosa para proyectos de aprendizaje automático, ciertamente no es la única para el trabajo. Hay una gran cantidad de bibliotecas disponibles en diferentes lenguajes de programación con diferentes soportes.

Referencias

[1] Desde cero: Machine learning

https://medium.com/datos-y-ciencia/introduccion-al-machine-learning-una-gu%C3%ADa-desde-cero-b696a2ead359

[2] MlLib

https://www.datahack.es/mllib-machine-learning-spark/#:~:text=MLlib%20o%20Spark%20MLlib%20es,de%20algoritmos%20de%20Machine%20Learning.

[3] Características MlLib

https://www.codetd.com/es/article/10369560

[4] Aprendizaje automático:

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

[5] Ejemplo – Algoritmo

https://www.baeldung.com/spark-mlib-machine-learning

Complementario

[1] Qué es el Machine Learning

https://www.youtube.com/watch?v=HHqlEnoGk54

[2] MLilb – tutorial

https://www.youtube.com/watch?v=d68VGJ7yAko