CAMBRIDGE UNIVERSITY PRESS

Diseño de algoritmos de machine learning con Spark
Escalabilidad

Resultados de aprendizaje

- Aprender a analizar la escalabilidad de una solución Big Data

Recap

- Aprendimos dos enfoques para diseñar soluciones de Big Data:
  - Local
  - Global
- ¿Cual es mejor?
  - La respuesta depende de varios factores, ya que tienen diferentes ventajas y desventajas
- Sin embargo, no hemos probado si son escalables o no, es decir, si son capaces de abordar conjuntos de datos arbitrariamente grandes

Objetivos de la sesión

- Medidas de escalabilidad
  - *Speed-up*
  - *Size-up*
  - *Scale-up*
- Cómo prepararlo todo para medir el tiempo de ejecución
- Probemos el arbol de decisión Local

Escalabilidad

- La escalabilidad de una solución de Big Data dependerá de varios factores relacionados con **los 'sobrecostes' incurridos**
  - ¿Tenemos mucho **tráfico** a través de **red**?
  - ¿Hemos **almacenado en caché** adecuadamente los datos que vamos a reutilizar?
  - ¿Hemos iniciado demasiados **trabajos Spark**?
  - …
- Para medir la escalabilidad de una solución de Big Data, tenemos que **probarla empíricamente** en diferentes escenarios con **diferentes tamaños de datos** y **disponibilidad de recursos**
- Las métricas de escalabilidad en computación distribuida son un campo muy estudiado. Aquí consideramos tres métricas clásicas que son más apropiadas para el análisis de datos a gran escala

Speed-up

- *¿Cuánto más rápido se pueden procesar los mismos datos con n núcleos en lugar de 1 núcleo?*

$$\text{Speed} - \text{up}(n) = \frac{\text{runtime in 1 core}}{\text{runtime in } n \text{ cores}}$$

- **El tamaño** de los datos de entrada **sigue siendo el mismo** independientemente de los núcleos utilizados
- Variamos $\boldsymbol{n}$ **desde 1 hasta el mayor número de núcleos** que tengamos
- Un **speed-up lineal** implicaría $\textbf{Speed} - \textbf{up}(\boldsymbol{n}) = \boldsymbol{n}$
  - En la práctica, es difícil lograr una aceleración lineal debido a los sobrecostes de comunicación y sincronización.

Speed-up

- Cuando el tamaño del conjunto de datos no permite ejecutar una versión secuencial del método, podemos **utilizar una mayor cantidad de núcleos como base** :

$$\text{Speed} - \text{up}(n) = \frac{\text{runtime in } b \text{ cores}}{\text{runtime in } b \cdot n \text{ cores}}$$

- O utilizar **un subconjunto de los datos originales**
  - El tamaño de los datos debe ser lo suficientemente grande para garantizar que los sobrecostes no limiten el estudio
- Nuestro análisis puede ser más sólido si lo hacemos para diferentes tamaños de datos.

Size-up

- *¿Cuánto tiempo se tarda en procesar un conjunto de datos* 1 *veces más grande?*

$$\text{Size} - \text{up}(n) = \frac{\text{runtime to process } n \cdot data}{\text{runtime to process } data}$$

- El número de **núcleos** se mantiene **fijo**
  - Sólo nos interesa medir el **impacto del tamaño de los datos**
- Creamos **subconjuntos** de los datos originales, p.ej., 10%, 20%,...,100%
  - Podríamos tomar el 10% como el más pequeño
  - **Nota**: La disponibilidad de recursos no debería limitar el cálculo de la métrica
- Un **aumento de tamaño lineal ->** $\boldsymbol{\text{Size} - \text{up}(n) = n}$, variando n
  - En la práctica, muy pocos algoritmos son lineales con respecto al tamaño de los datos

Scale-up

- Mide la capacidad de un sistema para ejecutar un conjunto de datos n veces más grande con un sistema n veces más grande.

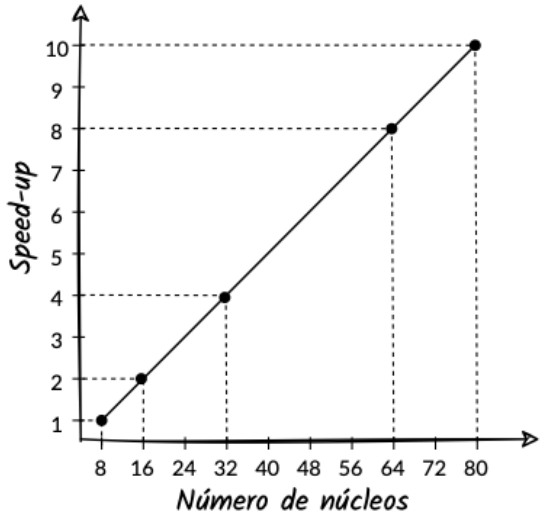
$$\text{Scale} - \text{up}(n) = \frac{\text{runtime to process } n \cdot data \text{ in 1 core}}{\text{runtime to process } n \cdot data \text{ in } n \cdot \text{cores}}$$

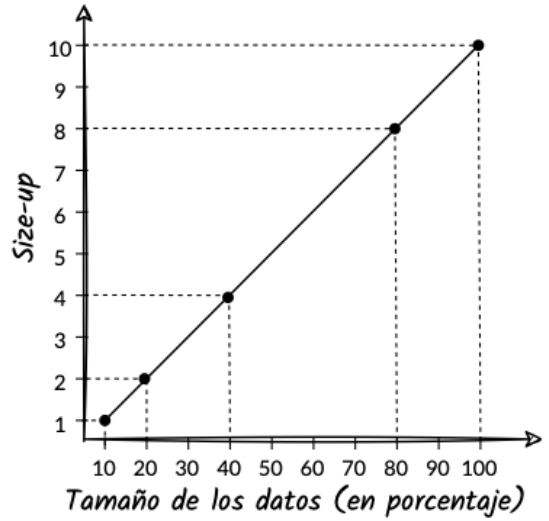
- Compara la **versión secuencial con** ejecuciones múltiples en las que **aumentamos tanto el tamaño de los datos como los núcleos** proporcionalmente
- **El scale-up perfecto sería 1;** pocas implementaciones realmente lograrían eso
- Al igual que con la aceleración, si no es posible ejecutar la versión secuencial, podemos establecer una base factible (p.ej., 8 núcleos y 10 % de datos)

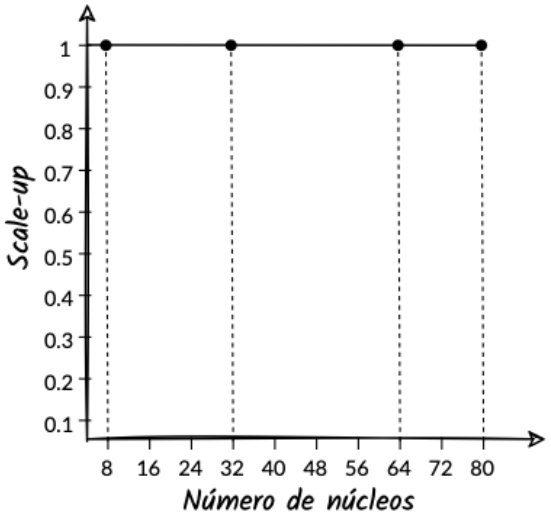
Medidas de escalabilidad: resumen

- Tiempo de ejecución hipotético logrado por un algoritmo escalable linealmente

Dataset size	8 cores	16 cores	32 cores	64 cores	80 cores
10%	10	5	2.5	1.25	1
20%	20	10	5	2.5	2
40%	40	20	10	5	4
80%	80	40	20	10	8
100%	100	50	25	12.5	10







Mide el tiempo de ejecución bien

- Para analizar speed-up, size-up and scale-up, necesitamos modificar la cantidad de núcleos y el tamaño del conjunto de datos
  - **Advertencia** : con una sola máquina seremos demasiado optimistas, ya que no consideraríamos la red
- Necesitamos ejecutar cada prueba/experimento de forma independiente, iniciando su propia SparkSession
- Tenemos que crear un programa independiente (en nuestro caso, un programa Python, con extensión .py)

```
def main(argv):
    # Main body of the program
    if __name__ == "__main__":
        main(sys.argv[1:])
```

Mide el tiempo de ejecución bien

- Llamamos al archivo '`train.py`'
- Podríamos ejecutar un programa independiente como:

```
>>> python train.py
```

- ¡Podemos ejecutar un script de Python en un cuaderno Jupyter !
- Podemos usar ' argv ' para que ese programa tome parámetros (número de núcleos y tamaño de los datos)

```
def main(argv):
    cores = int(argv[0])
    percentage = int(argv[1])
    ...
```

Mide el tiempo de ejecución bien

- Ahora podemos inicializar la SparkSession según la cantidad de núcleos:

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as sql_f
spark = (
    SparkSession.builder.master(f"local[{cores}]")
        .appName(f"Local DT con {cores} particiones")
        .config("spark.driver.memory", "16g")
        .config("spark.executor.memory", "16g")
        .getOrCreate()
)

sc = spark.sparkContext
```

Mide el tiempo de ejecución bien

- **Algunos trucos** para medir el tiempo de ejecución de manera justa
  - **No mires el 'Wall Time'** de ejecución del script
    - Consideraría el tiempo de ejecución de SparkSession, la lectura del conjunto de datos o cualquier otro procesamiento común
  - **Mide** únicamente el tiempo de ejecución **dentro del programa**
  - **Tenga cuidado con las operaciones vagas**
    - Splitting y repartitioning (lo hacemos como un truco) son operaciones vagas, necesitamos forzar una acción para que ignore ese tiempo
  - Es posible que quieras investigar la sobrecarga causada por Spark
  - Al trabajar con tu ordenador, debes tener en cuenta la influencia de otros procesos; sugerimos tomar el promedio de varias ejecuciones.

A Test on Local Decision Trees

- Vamos a **poner manos a la obra** e implementar nuestro primer programa independiente para ejecutar la solución local que diseñamos para DecisionTrees.
  - El script completo está disponible en el material complementario en:
    - at `'scalability-localDT.py'`
  - El script generará directamente el tiempo de ejecución, y el tiempo de ejecución sin tener en cuenta el sobrecoste de Spark

Hipótesis iniciales

- Analicemos la complejidad de los Árboles de Decisión **Locales**
- Su complejidad es $O(n \cdot \log n \cdot d)$, donde $n$es el número de instancias y $d$el número de características.
  - La cantidad de instancias importa y **no se puede** esperar que su tiempo de ejecución sea **lineal.**
    - Por ejemplo, si 1000 instancias toman 100 segundos, para 2000 instancias, no podemos esperar que sean 200 segundos, sino más
- ¿Qué podríamos esperar con las métricas que presentamos anteriormente?

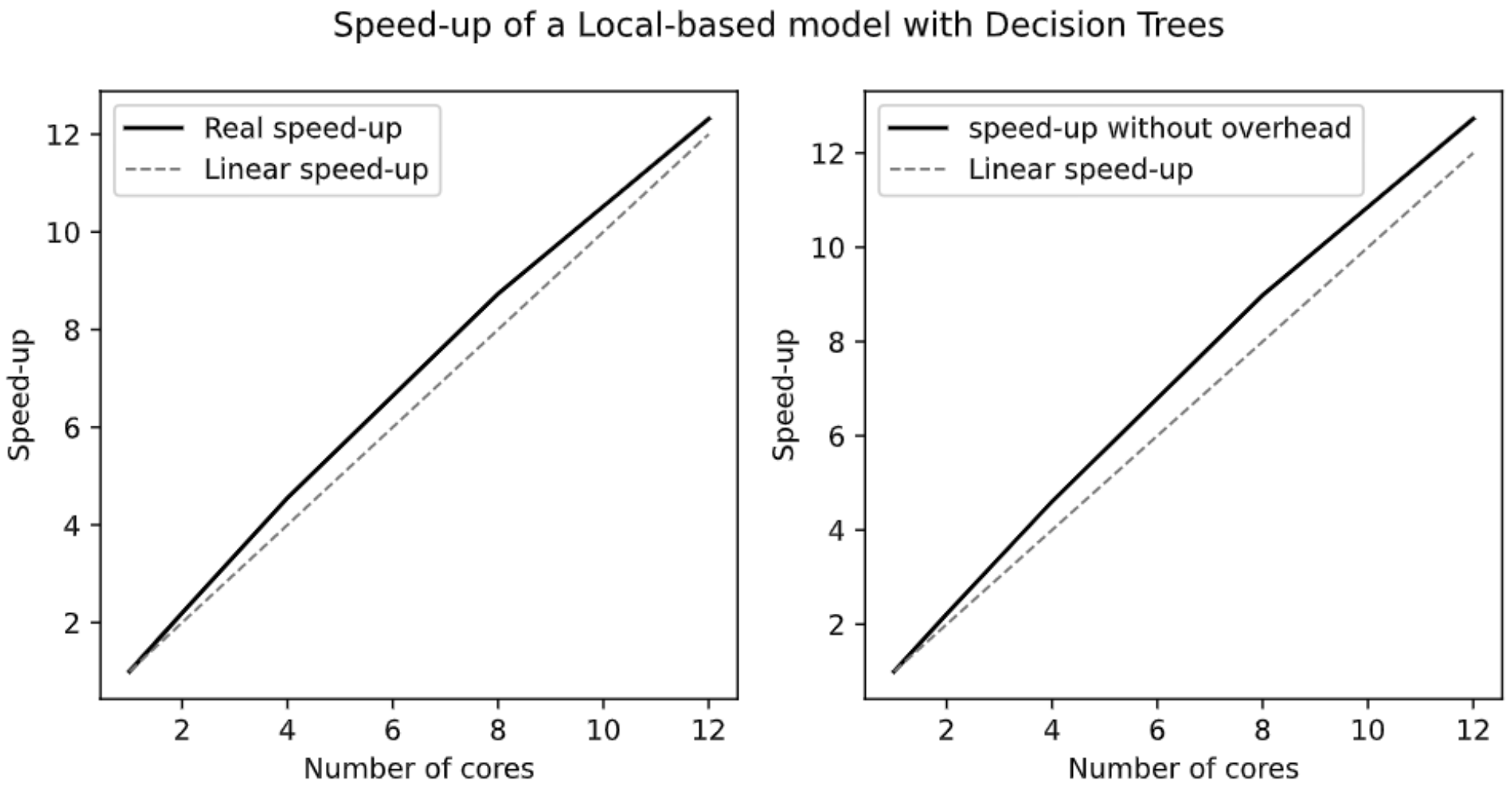
Hipótesis iniciales

- **Speed-up**
  - Debería ser **superlineal**, mejor que lineal, ¿por qué?
    - **Aumentar la cantidad de núcleos significa más particiones** para el enfoque local. Si reducimos la cantidad de instancias en cada partición, aprender cada árbol de decisiones sería más rápido
- **Size-up**
  - La complejidad no lineal de los árboles de decisión podría impedirnos obtener una mejora lineal
- **Scale-up**
  - Changing both cores and dataset size proportionally could compensate one another. We may achieve a linear behavior

Probando árboles de decisión locales

- Los siguientes experimentos se han ejecutado en un AMD Ryzen 9 5950X 16x3.4 GHz en el sistema operativo Linux Fedora 35
- Investigamos la aceleración con `[1, 2, 4, 8, 12]` núcleos. El ordenador tiene hasta 16 núcleos.
- Para size-up, analizamos `[10, 20, 40, 80, 100]` porcentajes de los datos
- Para scale-up, variamos la cantidad de núcleos `[1, 2, 4, 8, 10]` y el tamaño del conjunto de datos como `[10, 20, 40, 80, 100]`

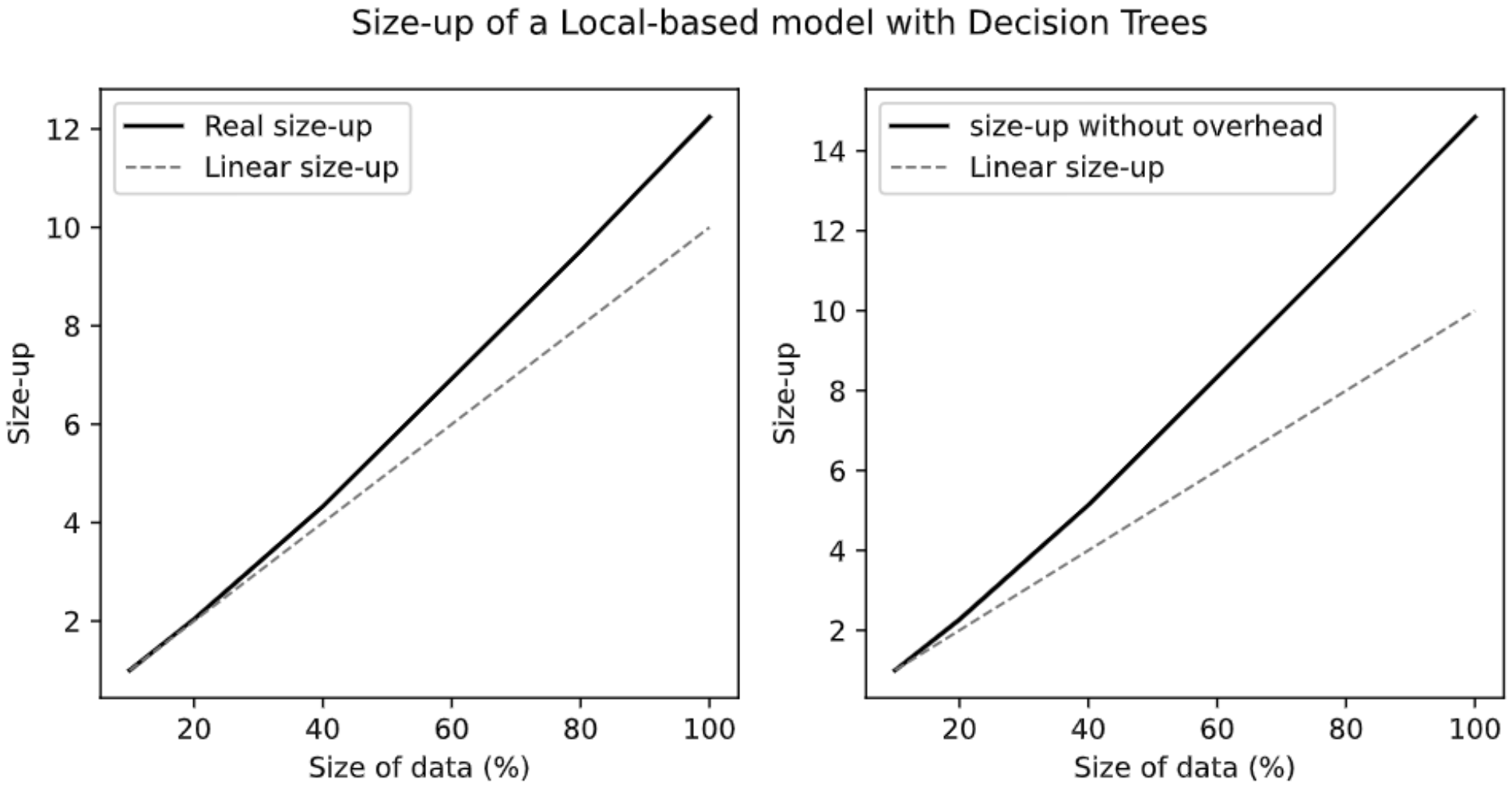
Probando árboles de decisión locales



Probando árboles de decisión locales: speed-up

- **Speed-up**
  - La aceleración real no es realmente superlineal. Si bien es mejor que la lineal cuando no se considera la sobrecarga de Spark, habríamos esperado que fuera una aceleración mucho mejor
  - Hay otros factores a tener en cuenta
    - Ejecutamos esto sobre un sistema operativo en modo pseudodistribuido
    - La sobrecarga causada por Spark se vuelve mayor en proporción al tiempo de ejecución total con una mayor cantidad de núcleos

Probando árboles de decisión locales: size-up

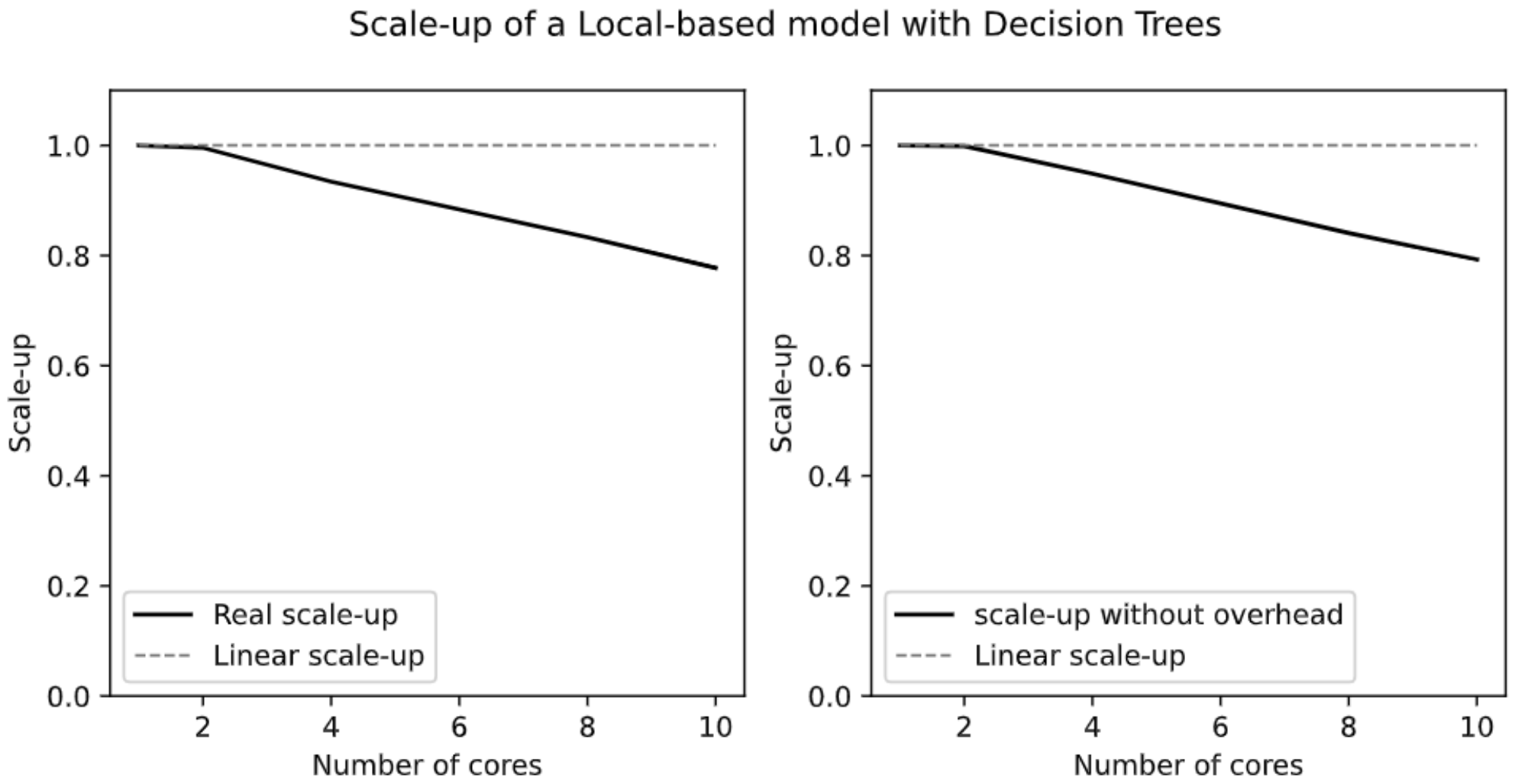


Probando árboles de decisión locales: size-up

- **Size-up**
  - Una vez más, no obtuvimos los resultados que esperábamos.
  - El size-up es peor que lineal, lo cual era de esperar debido a la complejidad de los árboles de decisión
  - Sin embargo, el tamaño del tiempo de ejecución total es mejor de lo que esperábamos.
    - El porcentaje de sobrecoste para nuestro baseline (10%) es proporcionalmente demasiado alto, lo que sesga el análisis (casi el 20% del tiempo de ejecución).

- ¡El baseline usado para calcular las métricas de escalabilidad puede tener una gran influencia!

Probando árboles de decisión locales: Scale-up

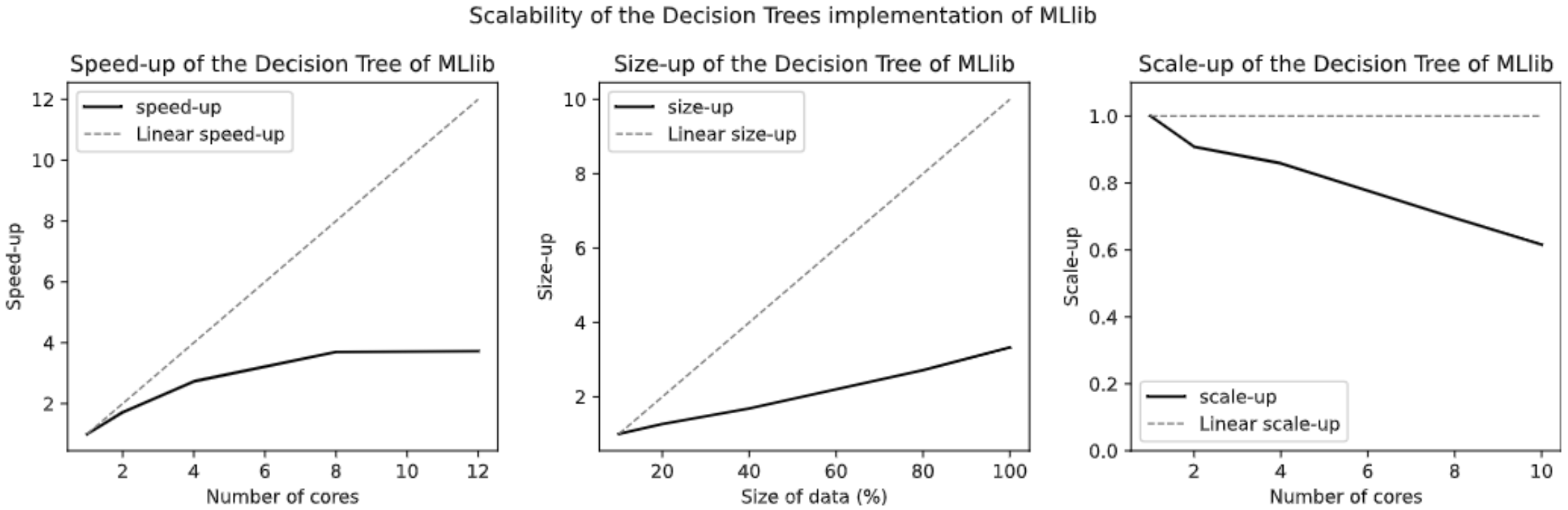


Probando árboles de decisión locales: Scale-up

- **Scale-up**
  - Nuevamente, no es lo que esperábamos, incluso sin tener en cuenta los sobrecostes
  - Los sistemas operativos pueden tener influencia
  - Es posible que Spark no pueda explotar todo el potencial del ordenador o que haya otros procesos que compitan por los recursos

Una prueba sobre árboles de decisión de MLlib

- Estblecemos `maxDepth` a 10
- Script disponible: `plot-scalability-global.py`



Árboles de decisión globales y locales: análisis de escalabilidad

- Nuestro análisis de escalabilidad (con sus propias limitaciones) muestra que el modelo local es mucho mejor que su contraparte global
  - Ten en cuenta que hemos utilizado un conjunto de datos específico, en un PC específico, lo que puede influir en gran medida en los resultados que presentamos aquí
- La aceleración está lejos de ser lineal para el árbol de decisiones global de MLlib, posiblemente causada por la sobrecarga de Spark

Take-Home Message

- Hemos introducido varias métricas para evaluar sistemas distribuidos
- La teoría y la realidad pueden diferir bastante debido a la complejidad del sistema distribuido subyacente
- Deberíamos establecer una base significativa y debería haber recursos suficientes para los experimentos planificados
- Medir el tiempo de ejecución no es trivial

- What's next?
  - Ensembles con Spark

¿Quieres aprender más?

- **Scalability of distributed systems**
  - Parallel database systems (DeWitt, 1992)
  - Evaluating the scalability of distributed systems (Jogalekar, 2000)
  - Performance evaluation of Classification algorithms in Spark (Hai, 2017)

CAMBRIDGE UNIVERSITY PRESS

Diseño de algoritmos de machine learning con Spark
Escalabilidad