Big data en la cosecha del sector agropecuario de Buenos Aires

Javier Ignacio Godoy

Universidad Nacional de General Sarmiento

Resumen. Estudiamos y desarrollamos una herramienta para el procesamiento de grandes volúmenes de datos en la cosecha del sector del agropecuario de Buenos Aires, lo cual llevaremos a cabo con algoritmos basados en el paradigma Map Reduce.

1 Introducción

Big Data refiere a un conjunto tan grande de datos no estructurados que tanto su recolección como su procesamiento representan un desafío. Los actores que han hecho uso de Big Data han podido desarrollar ventajas competitivas y los países desarrollados están invirtiendo y comenzando a aprovechar su potencial –por ejemplo, el gobierno de Estados Unidos comprometió 200 millones de Dólares al "Big Data Research and Development Initiative" [1].

En Sudamérica su exploración ha sido limitada, y presenta grandes oportunidades, tanto para el sector público como para el privado, por el potencial de contar con información (Big Data procesada) para cualquier organización. Argentina cuenta con una oportunidad única para liderar a la región en la explotación de Big Data, y ya existen iniciativas en ese sentido como la Plataforma de Datos Geolocalizados de Agro de la Argentina –AgroBigData– del Programa de Ciencia de Datos de la Fundación Sadosky[2].

A través de un mejor análisis de Big Data, se abre la posibilidad de hacer grandes avances en muchas disciplinas científicas, aportes a la innovación y mejoras de productividad y competitividad.

En términos esquemáticos, la problemática Big Data puede dividirse entre los desafíos tecnológicos de infraestructura y servicios (software y hardware) por un lado, y la dimensión del análisis de Grandes Datos por el otro.

En el marco de la beca otorgada por el Consejo Interuniversitario Nacional (CIN), nosotros nos enfocaremos en el procesamiento de datos de la cosecha en el sector del agropecuario de Buenos Aires, lo cual llevaremos a cabo con algoritmos basados en el paradigma Map Reduce[3]. El problema a resolver se describe a continuación.

1.1 Descripción del problema

Actualmente existen algoritmos geográficos utilizados en herramientas informáticas de Sistemas de Información Geográfica (SIG) que permiten procesar y analizar datos obtenidos por monitores de rendimiento. Dichos monitores son dispositivos que se combinan con las máquinas cosechadoras y permiten recolectar información valiosa a medida que la cosechadora va levantando los productos (maíz, etc)[4]: por ejemplo, procesando estos datos, se puede conocer cuántos kg por hectárea se recolecta de un producto particular en un determinado campo (lo cual se denomina rendimiento o rinde; a lo largo del trabajo se utilizan ambos términos como sinónimos). Analizándolo para un único año, puede no ser muy útil y que no sea posible sacar conclusiones objetivas, puesto que no se puede asegurar que se repitan esos mismos resultados para años posteriores. Sin embargo, tener esta información recolectada por varios años para los mismos campos, con iguales y diferentes productos, permite ver una tendencia de los rendimientos por zonas, con lo cual se podría realizar una mejor planificación de la siembra para obtener mejores resultados. Un ejemplo podría ser utilizar menos semillas en zonas donde se conoce que el rendimiento es bueno. Los algoritmos actualmente utilizados para el procesamiento de esta información desafortunadamente no escalan: para un volumen de datos suficientemente grande --como los que se presentan en el caso del agro-- los procesos no pueden llevarse a cabo por las limitaciones de tiempo y espacio de procesamiento.

El objetivo del presente trabajo es aplicar técnicas de Big Data para sustituir estos procesos que en la actualidad se realizan con algoritmos tradicionales, con la idea de poder escalarlos. El alcance del trabajo se limita a la implementación de algoritmos Map Reduce y al desarrollo de una herramienta que nos permita visualizar los resultados obtenidos de dichos algoritmos. Cabe destacar que mostraremos los resultados de procesar los datos de un único monitor --el tamaño de los datos es limitado-- para poder comparar el resultado obtenido por nosotros con el resultado obtenido por una herramienta de SIG.

2 Descripción del entorno y algoritmos implementados

Las pruebas se realizaron en una única computadora con las siguientes especificaciones:

- Procesador intel i3
- 6 GB RAM
- Sistema Operativo Ubuntu 14.10 64 bits
- Oracle Java SDK 1.8.0_25
- Apache Hadoop 2.6

2.1 Algoritmos implementados

Para el procesamiento de los datos se han implementado una serie de algoritmos Map Reduce que se describen en breve. Por temas de claridad y síntesis, cuando se mencionen las coordenadas o celdas como claves, estas siempre van acompañadas por los datos del año y producto, aunque estos últimos no se aclaren. Las coordenadas son puntos geográficos definidos por longitud y latitud donde se registró datos, mientras que las celdas son posiciones (i,j) dentro de una grilla (matriz) que se genera a partir de dichas coordenadas. Su utilidad se desarrollará en la explicación del algoritmo que las genera. Se anticipa que la descripción de los algoritmos se basa exclusivamente en el paradigma MapReduce. Este es un modelo de programación para el procesamiento de datos. Si bien el modelo es simple, no es nada trivial plasmarlo en la usabilidad de programas. La implementación de este modelo que utilizamos es el framework Hadoop[3], el cual permite ejecutar programas MapReduce en varios lenguajes. Es importante destacar que los algoritmos MapReduce son fácilmente paralelizables, lo que permite realizar análisis de datos en gran escala con el único requisito de disponer suficientes máquinas. MapReduce funciona descomponiendo el procesamiento en dos etapas: 'map' y 'reduce'. Cada etapa trabaja con pares clave-valor como entrada y salida, de tipos libremente definibles por el programador.

PODA

Objetivo: Se necesita eliminar los outliers, es decir, aquellos puntos que tienen un valor demasiado alto o demasiado bajo respecto del total de los datos (en general, se eliminan los percentiles 15 y 90, esto es, el 15% de los más bajos, y luego el 10% de los más altos del resto).

Implementación: algoritmo MapReduce para ordenar los valores según el rinde y luego quitar cierto porcentaje de los valores más bajos y más altos.

El map selecciona las coordenadas (longitud-latitud, además de producto y año) y valor del rinde; emite como clave k2[rinde, producto, año] y como valor esos mismos datos además de las coordenadas, es decir, emite v2[coordenadas, producto, año, rinde]. Entonces el map sería:

map (k,v) → List(k2,v2)

donde el par (k,v) son la clave-valor del input del map que vienen por defecto; lo importante en este punto es que v será cada línea de los datasets como input, lo cual hay que procesar para emitir k2 y v2, cuyos datos se mencionaron anteriormente.

En la fase de shuffle & sort, se ordenan según el valor del rinde (uno de los datos de la clave k2) pero se agrupan estos valores según el resto de los datos de la clave k2, es decir, según producto y año, para que sus valores (v2) vayan a la misma lista sin mezclar productos y años diferentes.

El motivo por el cual la clave y valor comparten casi los mismos datos, es porque desde la clave puedo ordenar los valores según el rinde (para luego eliminar los más bajos y altos) pero agrupándolos por producto y año. De esta forma, para todos los

productos y años iguales, se tiene una lista con todos los datos que interesan para futuros procesamientos.

Reduce: recibe List((k2, list(v2)) donde list(v2) son los valores [coordenadas, producto, año, rinde] ordenados en forma ascendente según el rinde. Luego, se eliminan el 15 percentil menor y, a este resultado, el 90 percentil mayor para cada producto-año diferente. Finalmente, para cada v2, se emite (k3,v3), donde k3 son la coordenada, producto y año, y v3 el valor de rinde para ese punto. Por lo tanto, el reduce sería:

reduce (k2, list(v2)) → List(k3, v3)

cuyo output es un nuevo dataset que no contiene los outliers, sobre el cual se trabajará a continuación.

TRAZADO DE GRILLA

Objetivo: Se requiere hacer un mapeo de puntos individuales con un valor determinado de rinde a una grilla (con celdas de tamaño regular) que contenga todos los puntos, donde cada celda tendrá un valor de rinde equivalente al promedio del rinde de todos los puntos que caen en la misma celda. Tener una estructura de celdas permite un mejor análisis comparativo de los datos, ya que tener la información únicamente de puntos geográficos representa una dificultad puesto que sería necesario volver a recolectar información sobre esos mismos puntos en cosechas futuras. En cambio, al representarlo por celdas, estas siempre están determinadas por coordenadas geográficas fijas y el interés radica en los valores de los puntos que caen dentro de estas celdas.

Implementación: Algoritmo Map Reduce que transforma el output del punto anterior [List(coordenada, rinde)] en List(celda, promedioRinde), donde celda además de representar una ubicación en la grilla (c,f), mantiene el año y producto.

Map: Dado un ancho de celda fijo (a), para cada clave se emite su celda correspondiente con

columna = longitud / a .

fila = latitud / a .

y como valor el mismo rinde de dicho punto.

En la etapa de shuffle & sort se agrupa por celda y aquellas con igual celda van a parar al mismo reduce.

Reduce: recibe List(k1,List(v1)) donde k1 es la celda y v1 los valores de rinde de todos los puntos que se encuentran en esa celda. Como resultado se emite List(k1, v2), donde v2 es el valor de rinde para esa celda dado por

con n = *length(List(v1))* , i ∈ {0, 1, 2, …, *n-1*} .

$$v2 = \sum_{i=0}^{n-1} \frac{(List(v1)_i)}{n} \quad .$$

RELLENO DE "AGUJEROS"

Motivación: La recolección de datos por parte de los monitores (lugar físico que recorre para obtener datos), junto a los procesos implementados para la poda y creación de la grilla, puede generar celdas vacías, es decir, celdas de las cuales no poseemos datos, lo cual no necesariamente sea correcto, ya que podemos tener conocimiento de las celdas cercanas (que la "rodean") y podemos asumir que tienen un valor promedio a estas. Para aquellas celdas que pudieron quedar vacías debido a la poda (poseían valores extremos, demasiado bajo o alto), con este algoritmo volveremos a tener datos con valores que se asemejen mejor al resto. Las celdas con datos no se modifican, sólo se modifican aquellas celdas que no poseían datos.

Implementación: Algoritmo Map Reduce cuyo input es el output del paso anterior.

Map: Emite map(k1, v1) → List(k1, List(v1)), donde k1 es una celda y v1 el valor de su rinde. Cada celda reporta el valor de su rendimiento a sus vecinos. Se entiende como vecinos, en este caso, a las celdas inmediatamente superior e inferior en la misma columna, y a las celdas inmediatamente a su derecha e izquierda en la misma fila. Además, la celda actual se considera vecino de sí misma. Para este caso particular, reporta su valor de rinde como negativo para diferenciarlo del resto, por motivos que se explicarán en la etapa reduce. Entonces, cada celda reporta:

- <C,F> -> rinde*-1 (misma celda con rinde negativo)
- <C,F+1> -> rinde (vecino superior misma columna)
- <C,F-1> -> rinde (vecino inferior misma columna)
- <C+1,F> -> rinde (vecino derecha misma fila)
- <C-1,F> -> rinde (vecino izquierda misma fila)

con C = posición columna actual y F = posición fila actual.

Reduce: Recibe List(k1, List(v1)) con k1 = celda y List(v1) = lista de rinde de los vecinos de k1. Para cada celda que recibe, aquellas que posean uno de sus valores de rinde negativo, se emite la misma celda con este valor pero positivo (el real). De esta forma, no se modifican las celdas que ya poseen valores. Para aquellas celdas cuya lista de valores de rinde no contiene alguna negativa, se verifica que posea al menos 3 vecinos, lo que es igual a tener al menos 3 valores de rinde reportados. Únicamente en este caso se emite una nueva celda (anteriormente considerado "agujero" a rellenar) con un valor de rinde promedio al reportado por sus vecinos. Luego, el resultado es List(k1, v2): k1 es la misma celda que se recibe y v2 está dado por

v2 = -v1 si List(v1) contiene un $v1_i < 0$.

$v2 = \sum_{i=0}^{n-1} \frac{(v1_i)}{n}$, n = *length(List(v1))* , i ∈ {0, 1, 2, …, *n-1*} si y sólo si $n >= 3$ y *cada* $v1_i >= 0$.

SUAVIZADO GAUSSIANO

Motivación: Se eliminan los valores que no son representativos, reemplazándolos por un valor que tenga en cuenta el valor de la celda actual y sus vecinos, teniendo en cuenta que mientras más cercano esté, mayor peso se le dará. A diferencia del algoritmo anterior, acá siempre se modifican los valores de las celdas, tenga información en las celdas o no ("agujeros", vecinos cercanos).

Implementación: Se recibe el output del paso anterior.

Map: Cada celda aporta cierto "peso" del valor de su rinde a las celdas cercanas (vecinos). Mientras más cerca se esté de la celda actual, mayor peso se le dará, teniendo el mayor peso el valor de la misma celda. Para esto se usa una matriz[5] (kernel) de 5x5, donde cada casillero de esta matriz tiene un peso fijo, que son el valor proporcional que se dará al valor de la celda actual (centro del kernel).

Entonces, cada celda reporta el valor de su rinde a sus vecinos (hasta 2 celdas de distancia), cuyo valor está determinado por el kernel.

Cada celda (c,f), con c = columna, f = fila se reporta a sus vecinos (y a sí misma) de la siguiente manera:

Sea *n* el tamaño del kernel y *m* el centro del mismo. Entonces para k = kernel[5][5], n = 5, m = 2.

Sean posi, posj las posiciones de fila y columna de k respectivamente, con i = 0, j = 0 contadores

```
mientras ( i < n)
        mientras ( j < n)
                posi = i - m;
                posj = j - m;
                v1 = rinde(celda(c,f)) * kernel[i][j];
                k1 = celda(c+posj, f+posi);
                emito (k1, v1);
                i++, j++;
        fin
fin
```

Los valores del kernel utilizado son:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Reduce: Recibe para cada celda (k1), todos los valores de rinde (v1) que le fueron "reportados" por sus vecinos: (k1, List(v1)). Nótese que se van a generar nuevos k1 (hasta el momento sin datos) que existen por ser vecinos de celdas con datos. Luego, el resultado final a emitir es (k1,v2) donde k1 es una celda (con la metadata del año y producto) y v2 será:

$$v2 = \sum_{i=0}^{n-1} \frac{(v1_i)}{\partial}\ , \mathrm{n} = length(List(v1)), \mathrm{i}\ \in\ \{0, 1, \dots, n-1\},$$
$$\partial = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} K[i][j], K \in N^{mxn}\ .$$

3 Resultados

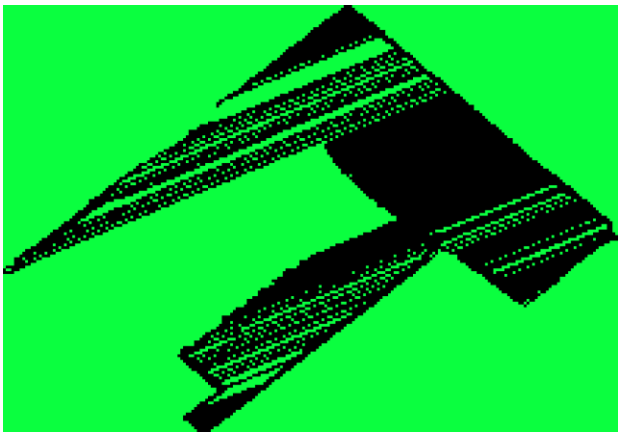
Se han implementado algoritmos geográficos diferentes a los tradicionales para analizar grandes volúmenes de datos haciendo uso de técnicas de Big Data. Para tener una imagen concreta de los resultados, se ha desarrollado una herramienta de visualización que procesa el resultado de los algoritmos implementados.

Contamos con dos tipos de gráficos:

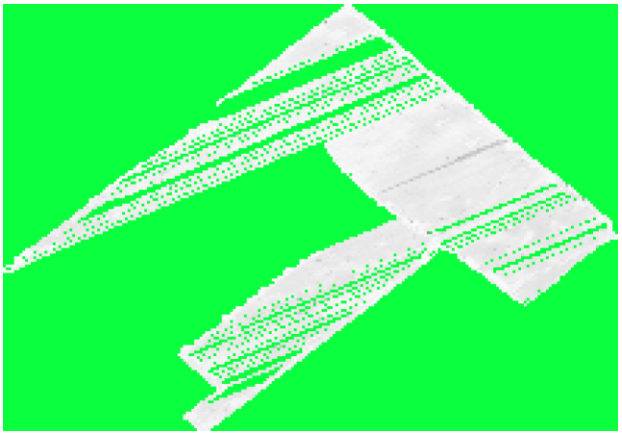
- Binario: aquellas celdas que contienen al menos un punto (independientemente del valor de rendimiento) se plasman en color negro, mientras que las otras celdas se presentan de color verde.
- De rendimiento: al igual que en el gráfico binario, las celdas sin puntos se presentan en color verde. Las celdas con datos tienen una escala de grises que va de blanco (el rendimiento es cero en esa celda) a negro (mayor rendimiento).

Cabe aclarar que las referencias de estos colores se aplican para todos los gráficos. A continuación presentamos los gráficos obtenidos luego del trazado de la grilla (a partir de los puntos geográficos), siempre para el mismo monitor.

Primeramente contrastamos el gráfico binario con el de rendimiento.

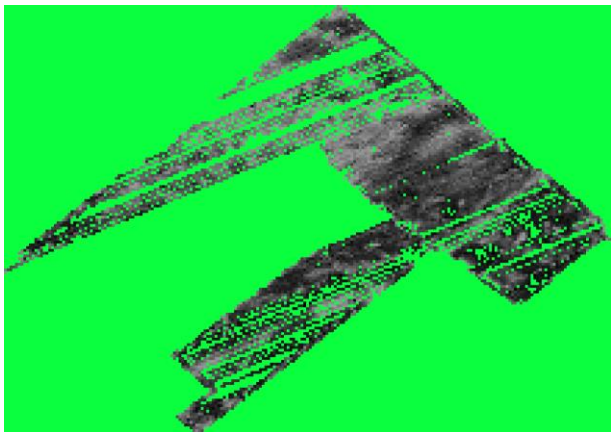


Binario



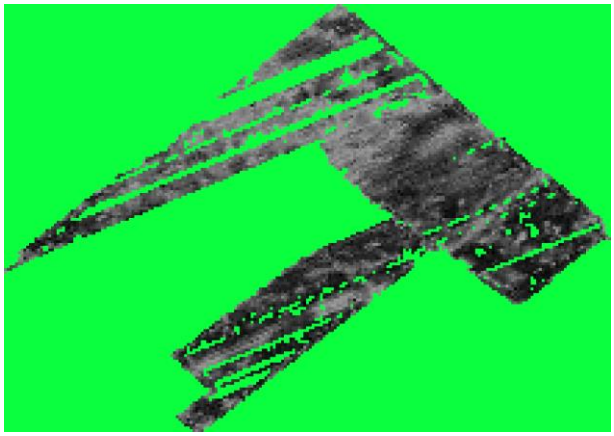
de rendimiento

Como se puede observar, el gráfico de rendimiento no refleja los datos de manera visualmente útil: pareciera que el rendimiento de casi toda la superficie del terreno con datos es muy bajo. Sin embargo, esto se debe a la presencia de valores extremos. Aplicando el algoritmo de poda o eliminación de outliers, se puede apreciar un mejor resultado:



de rendimiento con poda

Claramente la eliminación de valores extremos que no representan la mayoría de los valores permite obtener un resultado más útil. Sin embargo, es necesario aplicar un relleno de “agujeros” por las razones anteriormente descritas. El gráfico aplicando este tercer algoritmo se puede observar a continuación:

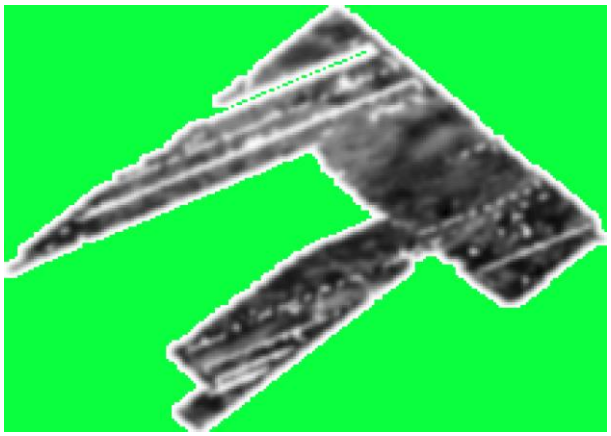


de rendimiento con poda y relleno de agujeros

Si bien se puede apreciar que algunas celdas se completaron, es evidente la presencia de ciertos huecos que claramente deberían ser celdas con datos.

Una implementación más eficiente que la anterior, que resuelve el problema del llenado de huecos, es el algoritmo del filtro gaussiano; al haberse realizado con un rango mayor de celdas vecinas respetando una distribución normal en los valores, logra un mejor suavizado de los datos y rellena la mayoría de los huecos existentes. El costo que “pagamos” al aplicar este algoritmo es un efecto difuminado que expande

los bordes de la grilla. El resultado de sumar este algoritmo a los anteriores, se puede visualizar en el siguiente gráfico:



de rendimiento con poda, relleno de agujeros y suavizado

4 Conclusiones y continuación

El principal resultado de este trabajo es el prototipo de una herramienta para procesar grandes volúmenes de datos del sector agropecuario de Buenos Aires. Esto, por un lado, permitirá poder predecir la cosecha esperada en cada región de la superficie estudiada, y de ahí sacar provecho económico de esta información. Asimismo, esta herramienta podrá ser usada en otras regiones y a la vez contemplamos la posibilidad de aplicarla a otros campos en los que se manejen grandes volúmenes de datos y este tipo de estudio estadístico tenga relevancia.

El objetivo a corto plazo son ciertas modificaciones en la implementación actual: crear una nueva implementación para obtener el valor del rinde de acuerdo a la masa recolectada por la cosechadora y trabajar con una proyección de las coordenadas geográficas, puesto que las distancias medidas en latitud y longitud varían dependiendo de la ubicación en la Tierra (por ejemplo un grado de latitud en el ecuador no mide lo mismo en metros que un grado cerca de los polos, debido a que la Tierra no es una esfera perfecta, sino que los polos son más achatados).

El objetivo a mediano plazo es migrar esta implementación a un cluster para procesar grandes volúmenes de datos de la cosecha. En la agenda de trabajo próximo está eliminar el sorting de los datos y reemplazarlo por un algoritmo de búsqueda de percentiles (usado en la poda) --que en general puede hacerse en O(n)--, tratando minimizar la comunicación entre las máquinas del cluster que usaremos; eventualmente, podemos relajar el requerimiento de percentiles exactos, y utilizar una aproximación de manera más rápida.

Referencias

1. Tom Kalil. Big Data is a Big Deal. Office of Science and Technology Policy (2012) http://www.whitehouse.gov/blog/2012/03/29/big-data-big-deal.
2. E. Feuerstein. El Programa de Ciencia de Datos de la Fundación Sadosky (2014).
3. Tom Whit. Hadoop: The Definitive Guide. O'Relly Media, 2nd edition (2012).
4. http://en.wikipedia.org/wiki/Grain_yield_monitor.
5. http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm.