Informe del TP: Fine Foods Review

Martín Queija, *Padrón Nro. 96.455*
tinqueija@gmail.com

Estanislao Ledesma, *Padrón Nro. 96.622*
estanislaoledesma@gmail.com

Martín Bosch, *Padrón Nro. 96.749*
martinbosch17@gmail.com

Grupo en kaggle: fsociety

2do. Cuatrimestre de 2016
Organización de Datos
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen
Aproximación por regresión

Índice

1. Introducción

El objetivo de nuestro trabajo práctico fue indicado especificamente por Luis Argerich. Como utilizariamos el framework de procesamiento paralelo Spark Hadoop, se nos fue instruido una nueva orientacion: Intentar predecir la mayor cantidad de reseñas de comidas que fuera posible, optimizando la cantidad de tiempo computo e intentando obtener el menor error posible.

2. Cluster Computing

El framework Apache Spark es utilizado para paralelizar tareas. Como estas son asociativas y commutativas se pueden distribuir sobre muchos servidores. En un primer intento configuramos un cluster de 2 computadoras. Sin embargo la capacidad de procesamiento de las computadoras disponibles para el cluster eran notablemente desigual. Con una simple corrida de prueba nos dimos cuenta de la ineficiencia del cluster del que disponiamos. El resultado fue notablemente superior al paralelizar las tareas sobre los 8 nucleos de la computadora.

3. Set de datos

El set de datos que utilizamos en nuestro trabajo practico fue cortesia de Julian McAuley, de la Universidad de San Diego en California, Estados Unidos.

El data set consta de 142.8 millones de reviews que datan desde mayo 1996 hasta july 2014. Las reviews pertenecen a todos los tipos de items pertenecientes a Amazon que puedan ser reseñados. Una vez filtrados items duplicados, ya que pueden que las reviews se repliquen automaticamente para productos casi identicos, quedan 83.68 millones de reviews.

3.1. Preprocesamiento de los datos

En una primera instancia se extrajo la puntuación de los datos crudos (utilizando parserGRANDATASET.py) para no desperdiciar *features* con símbolos que a la escencia de la implementación no hacen de utilidad. De todos los features elegimos quedarnos unicamente con el ReviewID, su texto asociado y el score de la misma.

3.2. Training y Learning Set

El set de datos descargado: itemdedup.json.gz fue inicialmente dividido en un set de entrenamiento de 1 millon de reviews y otro set de test de 82.68 millones de reviews. Al realizar la primera pasada por sobre el set de datos con un millon de reviews para entrenamiento el diccionario de las claves para cada banda se hizo demasiado grande como para ser manejado en la memoria disponible (8gb RAM) y la ejecucion debio finalizar. Lo mismo sucedio con 500 mil reviews para entrenamiento, y finalmente pudimos establecer un valor adecuado a la memoria de 300mil reviews para entrenamiento. De esta manera contabamos con un set de entrenamiento de 300mil reviews, para predecir unas 82.68 millones de reviews.

Como el tiempo aproximado de corrida por sobre todo el set de dato costaba aproximadamente 14hs, se utilizo un set de test reducido (1 millon de reviews) para probar el desempenio del algoritmo con distintos valores de hyperparametros.

4. Machine Learning

Para alcanzar el objetivo, la intención es diseñar un algoritmo que *aprenda* a partir de un set de reseñas ya puntuadas para así poder predecir la calificación de nuevas reseñas. Parte del set de reseñas ya puntuadas se reserva para probar el "aprendisaje" del algortmo. De esta manera se puede determinar la presicion del algorítmo al predecir reseñas y comparar la predicción con el valor esperado. La comparacion utilizada se denomina Mean Squared Error y se calcula sumatoria total de los errores al cuadrado sobre el total de puntos. Esta relación proporciona un grado de confianza sobre nuestro algorítmo para predecir puntuaciones de nuevas reseñas.

4.1. KNN

La base del diseño se basa en el método de reconocimiento de patrones llamado KNN (K-Nearest-Neighbour). Este método es aplicable para realizar regresión o clasificación. El set de datos provisto califica las reseñas con enteros del 1 al 5. Sin embargo el enunciado afirma que nuestro algortímo puede predecir

valores no enteros en el intervalo [1,5]. Por lo tanto establecemos que nuestro método de predicción sera regresivo.

El mecanismo de KNN consiste en encontrar los K-Vecinos mas cercanos a la nueva instancia de datos que queremos clasificar. Para clasificación se predice la clase mayoritaria entre los K-Vecinos. Para regresión se calcula el promedio.

El hyperparámetro K se deduce a partir de una grid search, es decir, a partir de prueba y error probaremos diferentes valores de K para encontrar el que mejor ajuste a nuestro set de datos.

4.2. Locality Sensitive Hashing

Para evitar calcular por fueza bruta las distancias al aplicar KNN, utilizaremos LSH. Entonces dado el texto que queremos clasificar aplicamos una funcion de hashing y accedemos al bucket de la tabla de hash señalado por la misma, promediamos los scores que se encuentran dentro de dicho bucket y utilizamos este promedio para la prediccion. Para reducir los falsos positivos (afectan la performance) utilizaremos mas de una funcion de hash o minihash, supongamos r funciones, obteniendolas de las familias universales de Carter-Wegman o las clasicas como Jenkins. Y para reducir los falsos negativos (afectan la precision) agruparemos estos minihash en b grupos, es decir, que vamos a tener b tablas de hash con T cantidad de buckets cada una. Los hiperparametros b, r y T seran determinados de la siguiente manera: primero se determinara el b mas grande posible en base a la memoria disponible, para asi tener mayor presicion. Luego determinaremos r y T utilizando una grid search teniendo en cuenta la performance del algoritmo.

5. Detalles del Algoritmo

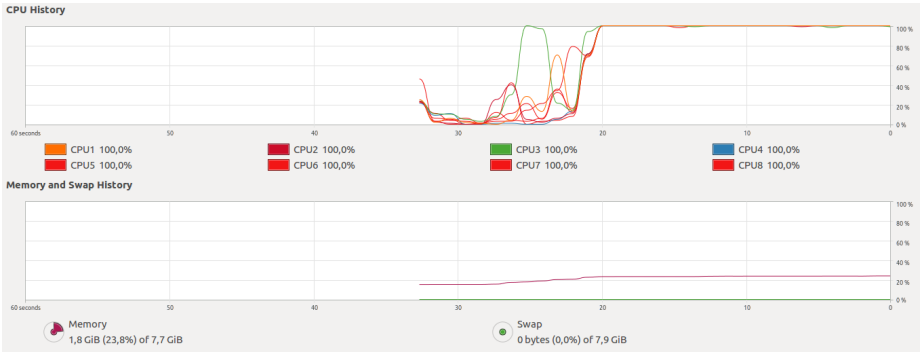
5.1. Spark Framework

El algoritmo fue desarollado en Python utilizando el framework Spark. El mismo presenta el paradigma Map-Reduce para implementar un metodo de procesamiento paralelo. A partir de los datos crudos se arma un RDD (Resilient Distributed Dataset). A estos se les puede aplicar una serie de operaciones para moldear el set de datos para extraer el analisis deseado. Las operaciones "Lazy", no se computan en el momento. Se almacenan en el RDD hasta que se llama a una operacion "Non Lazy". La operacion "Non Lazy" efectua sobre el RDD todas las operaciones "Lazy"que hayan sido almacenadas y devuelve un resultado.

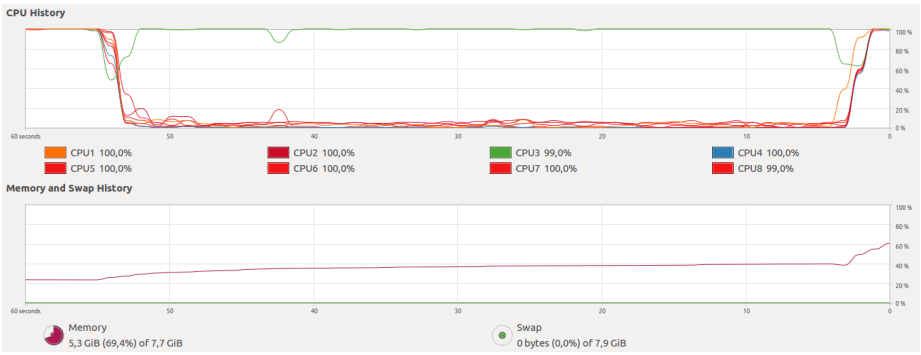
Al configurar distintas variables del entorno de spark, como el número de cores, de executor, la cantidad de memoria por executor o utilizar el serializador Kryo-Serializer, no obtuvimos una mejora en el rendimiento, por lo tanto dejamos los valores por default para correr nuestro algoritmo [9].

5.2. Implementacion

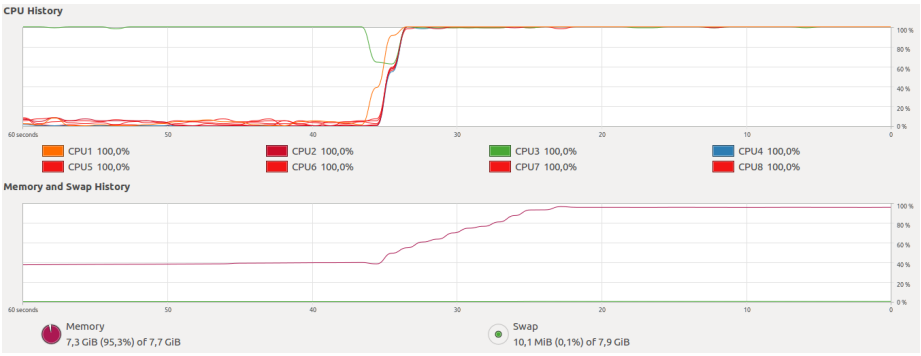
El algoritmo esta dividido en tres partes: Entrenamiento, prediccion y calculo del error cuadratico medio.



En la parte de entrenamiento se calculan los codigos de cada una de las bandas del review, esta operacion se realiza paralelamente sobre el set de datos. Se particionan los set de datos en 8 particiones tal que haya un worker por nucleo del microprocesador. La operacion Non Lazy es un collect(), en la que se descargan los datos de la memoria de los Workers a la memoria de ejecucion del thread del interprete de python.

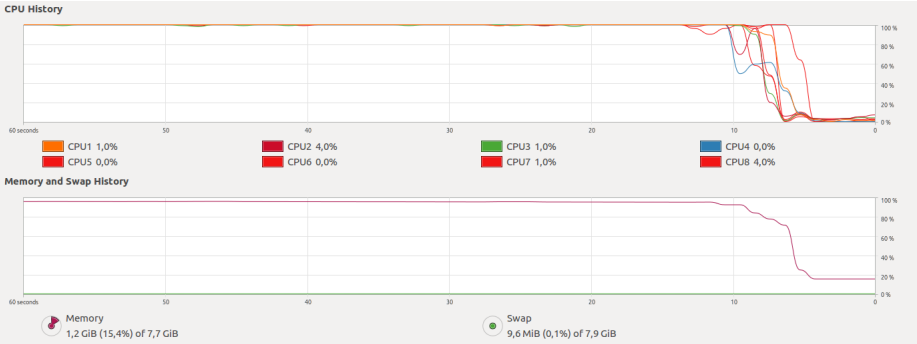


Con estos datos se construye un diccionario de python donde la clave es el codigo de hash asignado a cada banda y el valor una lista de todos los promedios que hay de entrenamiento para esa banda. Esta es la parte No Paralela del algoritmo. Una vez armado el diccionario se procede a predecir el set de datos.



Luego en la etapa de prediccion, se carga un RDD con el set a predecir y se mapea al set de datos las instrucciones para calcular el score de cada review consultando en el diccionario los valores aprendidos. Se nos presentaron casos en los que las bandas de los minhashes de algunos reviews no encontraban nin-

gun valor aprendido en el diccionario. En estos casos si estimaFaltantes = False, los reviews que no puedan ser calculados seran descartado. Si estimaFaltantes = True para todas las reviews se estima el valor medio de todos los valores posibles: 3. Es una aproximacion heuristica bastante pobre, pero se aplico con el fin de ver como afectaba el error cuadratico medio resultante.



La siguiente operacion Non Lazy depende del flujo del programa. Si escribeScoresKaggle = True, se realiza un collect() y cada prediccion se escribe a un archivo csv. En el caso contrario, se mapean adicionales operaciones para calcular el error cuadratico medio donde la operacion Non Lazy es un reduce que acumula la diferencia al cuadrado de predictedScore y actualScore, y a su vez (como se reduce una tupla) se acumulan la cantidad de reviews para los cuales hubo prediccion.

5.3. Funcionamiento

Los hyperparametros del programa se configuran en el archivo LSH.py, y para correr se utiliza un script que establece las configuraciones inciales para spark. El script es el siguiente:
./bin/spark-submit
–class org.apache.spark.examples.SparkPi
–master local[8]
/home/root/LSH.py

Configuracion spark de manera local en 8 nucleos

5.4. Funciones de hashing

Con un claro objetivo en mente: predecir la mayor cantidad de reviews en la menor cantidad de tiempo (tambien consideramos elocuente no reinventar la rueda), decidimos utilizar la funcion de hashing provista por python: hash(). Esta funcion provee baja probabilidad de colision con una alta velocidad de computacion. Como utilizamos b Bandas de n cantidad de hashes cada una necesitamos b * n cantidad de diferentes funciones de hashing. Una manera eficiente y veloz de obtener una aproximacion semejante es de computar una unica vez b * n numeros semi-aleatorios. Para obtener las diferentes funciones de hashing con estos numeros semi-aleatorios probamos XORear o sumar el valor de hash(). La implementacion del XOR presento menor costo computacional (traducido en menor velocidad de procesamiento) y a su vez mejoro el error

obtenido.

5.5. Variables e hyperparametros del algoritmo

maxint = sys.maxint
Se utiliza para la primera comparacion del minhash. De esta manera siempre el primer minhash comparado sera menor a maxint.

bandas = 15
Cantidad de bandas.

hashesPorBanda = 1
Cantidad de funciones de hash por banda. Encontramos que al aumentar la cantidad de hashes por banda aumentaba significativamente la cantidad de reviews carecientes de score. Esto se da porque al tener dos hashes por banda aumenta la probabilidad de que las bandas no coincidan ya que se calculan a partir de dos hashes.

cantHashes = bandas * hashesPorBanda
Cantidad de diferentes funciones de hashing. Encontramos que un valor superior a 15 generaba diccionarios demasiado grandes como para ser manejados en la memoria disponible

nGram = 8
Constante de los n-gramas. La implicancia directa de este hyperparametro se vio reflejada sobre la cantidad de reviews que el algoritmo no pudo predecir. Con valores menores comenzaba a incrementar el error cuadratico medio. Al aumentar la cantidad de caracteres de contexto a 13 o superior los contextos eran tan amplios que los minhashes comenzaban a diferir notablemente entre los reviews, causando una menor tasa de recall.

xorHash = True
Define si las variaciones de las funciones de hash se obtienen XOReando el numero random o sumandolo. A partir de investigacion se pudo concebir que la opcion mas acertada para obtener distintos valores de hash era aplicando la operacion XOR. Se pudo notar una leve mejoria en el error y a su vez en el tiempo de procesamiento.

shingleaWords = False
Si shingleaWords == True, shinglea por palabras de lo contrario shinglea por caracter.

escribeScoresKaggle = False
Si el flujo del programa es escribir resultados para el set de datos de kaggle, o para el set de datos grande.

hashDisplaces = []
En este arreglo se almacenan los numeros random para generar las distintas funcinoes de hashing.

valoresPorBanda = dict()
En este diccionario se anotan los scores por clave de banda.

lineasEnLearn = 300000
Cantidad de reviews para entrenar el algoritmo.

cantReviewsToPredict = 82680000
Cantidad de reviews para predecir.

5.5.1. Estima Faltantes?

Si estimaFaltantes = True, aquellos reviews que no tuvieron ningun match para sus bandas se estiman con un valor de 3. De lo contrario se descartan.

estimaFaltantes = False
Reviews to learn: 300000
Reviews to predict: 600000
Bandas: 15
Hashes por banda: 1
Bandas calculadas: 4500015
Cantidad de hashes: 15
XOR: True
Escribe scores Kaggle: False
Shinglea por caracteres con K-Grams: 8
Claves en el diccionario: 563907
Predicted / To Predict: 0.999955000075
Reviews con score: 599974
Error cuadratico medio: 1.06924094114

estimaFaltantes = True
Reviews to learn: 300001
Reviews to predict: 600001
Bandas: 15
Hashes por banda: 1
Bandas calculadas: 4500015
Cantidad de hashes: 15
XOR: True

Escribe scores Kaggle: False
Shinglea por caracteres con K-Grams: 8
Claves en el diccionario: 572900
Predicted / To Predict: 1.0
Reviews con score: 600001
Error cuadratico medio: 1.06834047868

6. Resultados sobre el dataset

————-1————
Learn file: /parsedTrain100k.csv
Reviews to learn: 100000
Test file: /parsedTestFull.csv
Reviews to predict: 82680000
Bandas: 15
Hashes por banda: 1
Bandas calculadas: 4500000
Cantidad de hashes: 15
XOR: True
Escribe scores Kaggle: False
Shinglea por caracteres con K-Grams: 8

Claves en el diccionario: 1050230
Predicted / To Predict: 0.93688672
Reviews con score: 0.87797646
Error cuadratico medio: 1.46986627444
real 838m24.028s
user 12m1.468s
sys 1m56.160s

————-2————
Learn file: /parsedTrain300k.csv
Reviews to learn: 300000
Test file: /parsedTestFull.csv
Reviews to predict: 82680000
Bandas: 15
Hashes por banda: 1
Bandas calculadas: 4500000
Cantidad de hashes: 15
XOR: True
Escribe scores Kaggle: False
Shinglea por caracteres con K-Grams: 8

Claves en el diccionario: 2878527
Predicted / To Predict: 0.93688672
Reviews con score: 77461794

Error cuadratico medio: 1.37767813547
real 907m22.183s
user 16m48.616s
sys 3m13.928s

————-3————
Learn file: /parsedTrain300k.csv
Reviews to learn: 300000
Test file: /parsedTestFull.csv
Reviews to predict: 82680000
Bandas: 15
Hashes por banda: 1
Bandas calculadas: 4500000
Cantidad de hashes: 18
XOR: True
Escribe scores Kaggle: False
Shinglea por caracteres con K-Grams: 8

Claves en el diccionario: 2879600
Predicted / To Predict: 0.938506337687
Reviews con score: 77595704 Error cuadratico medio: 1.27984225047
real 1004m3.053s
user 25m5.672s
sys 1m86.264s

7. Otras implementaciones descartadas

7.1. Lideres y seguidores con Delta TFIDF

Otra opción que implementamos, en paralelo con LSH, fue aproximar KNN con lideres y seguidores, utilizando como métrica una variante de TF-IDF (con BM25 y normalizando los datos), llamada Delta TF-IDF [10] utilizada para el análisis de sentimiento, para lo cual armamos diccionarios de palabras positivas y negativas, en lo cuales contabamos la frecuencia de cada palabra en los reviews positivos y negativos, consideramos positivos a los reviews de score 4 y 5, y negativos a los de score 1, 2 y 3. En este caso al procesar los datos quitamos todos los signos, las stopwords y no utilizamos ngramas.

Descartamos esta idea porque era muy lenta al comparar el valor de delta TFIDF contra otro review, por lo tanto al comparar contra los lideres, y luego contra los seguidores del lider mas cercano se volvia muy lento el algoritmo al predecir, tardando en promedio 3 segundos por review.

Referencias

[1] Image-based recommendations on styles and substitutes. J. McAuley, C. Targett, J. Shi, A. van den Hengel. SIGIR, 2015

[2] Inferring networks of substitutable and complementary products. J. McAuley, R. Pandey, J. Leskovec. Knowledge Discovery and Data Mining, 2015

[3] LSH (Locality Sensitive Hashing, WikiPedia, https://en.wikipedia.org/wiki/Localitysensitive_hashing

[4] Data Paralelism , WikiPedia, https://en.wikipedia.org/wiki/Data_parallelism.

[5] KNN (K Vecinos mas cercanos), WikiPedia, https://es.wikipedia.org/wiki/Kvecinos_m %C3 %A1s_cercano

[6] Apache Spark, WikiPedia,https://en.wikipedia.org/wiki/Apache_Spark

[7] Programacion neurolinguistica, https://es.wikipedia.org/wikiProgramaci %C3 %B3n_neuroling %C3 %B

[8] Apunte oficial de la materia, Luis Argerich

[9] CONFIGURACION SPARK https://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/

[10] Delta TF-IDF http://ebiquity.umbc.edu/filedirectory/papers/446.pdf