Tópicos especiales en Telemática
Big Data
Similaridad de Textos

Juan David Pérez - Andrés Atehortúa
jperezp2@eafit.edu.co aatehor4@eafit.edu.co
Escuela de ingenierías
Universidad EAFIT

Abstract*— En este documento se muestra el reporte con respecto a la práctica de *Clústering de Documentos a partir de Métricas de Similitud basado en Big Data

keywords– HPC, Paralelización, Kmeans, Similaridad, Vectorización, Coseno, *Cluster*, Spark, MLlib

I. INTRODUCCIÓN

A partir de este documento se busca plantear los resultados del proyecto de *Clústering de Documentos a partir de Métricas de Similitud basado en Big Data*, donde se mostrará la implementación en spark de *Cosine similarity* como método de similaridad y kmeans para trabajar en la agrupación de documentos.

II. MARCO TEÓRICO

Este proyecto comprende varios aspectos, los cuales se integraron para llevar a cabo su desarrollo; uno de estos es un algoritmo de similaridad el cual se encargara de hallar la frecuencia con la cual se repiten ciertas palabras entre documentos. Para este caso se eligió *cosine similarity*, el cual es una medida entre 2 vectores en un espacio; dicha medida se obtiene al evaluar el coseno del ángulo entre dichos vectores. La función coseno esta dada por:

$$cos(\alpha) = \frac{A*B}{||A||*||B||}$$

Como se menciona anteriormente esta medida se aplica entre vectores para lo cual se deberá convertir los documentos en un tipo de matriz tf-idf (que despues se descompone en vectores), de tal manera que se pueda aplicar dicho algoritmo de similitud a partir de la frecuencia de cada palabra, para la modelación de un documento a un espacio vectorial lo primero que se hace es crear un diccionario con los términos presentes en el documento (eliminando las *stopwords*), posteriormente se utiliza la frecuencia de cada termino en el vocabulario que posee cada documento para la representación del mismo en el espacio vectorial, definición a partir de funciones

$$tf(t,d) = \sum_{x \in d} fr(x,t)$$

Donde d es el documento, t el número de veces que el termino esta en el documento y $fr(x,t)$ está definida como:

$$fr(x,t) = \begin{cases} 1 & \text{if } x = t \\ 0 & otherwise \end{cases}$$

Por lo tanto, la creación del vector que representa el documento está dada por

$$\vec{v_{d_n}} = (tf(t_1,d_n)), (tf(t_2,d_n)), ..., (tf(t_n,d_n))$$

Una vez se tiene el vector, el termino tf-idf (*term frequency – inverse document frequency*) busca solucionar el problema de que términos realmente le aportan al texto ya que en muchos casos los que más se repiten no son los que más aportan o son los más importantes. Para esto primero debemos normalizar el vector, a partir de la formula

$$\hat{v} = \frac{\vec{v}}{|||\vec{v}||_p}$$

Y luego aplicar el idf, el cual está definido como

$$idf(t) = log \frac{|D|}{1+|d:t \in d|}$$

Donde |D| es el espacio de documentos definido por D = d1,d2,...,dn, y d:t E d es el número de documentos donde el termino t aparece, cuando la función tf(t,d) <> 0, de este proceso sale un vector el cual se multiplica con una matriz identidad y posteriormente esta matriz se multiplica con una que sale en las primeras etapas del proceso con la frecuencia de los términos en cada documento. Después de esto se tiene una matriz tf-idf con el conjunto de documentos a ser comparados

luego de tener esto previamente calculado se procede a aplicar el algoritmo de k-means; este algoritmo es un método de cuantificación vectorial el cual tiene como objetivo dividir un conjunto de observaciones en n sub-grupos o *clusters*. Todo ello con el fin de agrupar los documentos que comparten una mayor similitud en n grupos, esto se puede llevar a aplicaciones de recomendaciones, buscadores, entre otras.

El algoritmo de k-means, requiere del número de *clusters* y un *RDD* con los valores resultantes del *cosine similarity*, primero el algoritmo establece unos puntos centrales para los k *clusters*, que pueden ser aleatorios o partiendo del conjunto de datos, el algoritmo opera en dos pasos básicamente, define los centros y posteriormente asigna los datos más cercanos a cada uno de estos puntos centrales.

III. Análisis y Diseño mediante Analítica de Datos (ETL-Procesamiento-Aplicación)

IV. Implementación

A. HPC - Paralelo

Vectorizer

```
def word_frequencies(word_vector):

    num_words = len(word_vector)
    frequencies = defaultdict(float)

    if rank == 0:
        data = word_vector
    else:
        data = None
    data = comm.bcast(data,root = 0)

    for word in data:
        frequencies[word] += 1.0/num_words
    return dict(frequencies)
```

K-means

```
def update_clusters(self):

        def closest_center_index(vector):
            similarity_to_vector = lambda center:
    similarity(center,vector)
            center = max(self.centers, key=
    similarity_to_vector)
            return self.centers.index(center)

        self.clusters = [[] for c in self.centers]

        if rank == 0:
            info = self.clusters
        else:
            info = None
        info = comm.bcast(info,root=0)
        pos = rank
        while pos < len(self.vectors):
            vector = (self.vectors)[pos]
            index = closest_center_index(vector)
            if vector != None:
                info[index].append(vector)
            pos += size
        comm.Barrier()
self.clusters = info
```

B. Apache Spark

main.py

```
rdd = sc.parallelize(dataset)
shemaData = rdd.map(lambda x: Row(num=x[0], title=x
    [1], text=x[2]))
dataFrame = sqlContext.createDataFrame(shemaData)
tokenizer = Tokenizer(inputCol="text", outputCol="
    words")
wordsData = tokenizer.transform(dataFrame)
hashingTF = HashingTF(inputCol="words", outputCol="
    rawFeatures")
featurizedData = hashingTF.transform(wordsData)
idf = IDF(inputCol="rawFeatures", outputCol="
    features")
idfModel = idf.fit(featurizedData)
rescaledData = idfModel.transform(featurizedData)
rescaledData.select("title", "features").show()
#Normalizacion y transformada de la matriz
normalizer = Normalizer(inputCol="features",
    outputCol="norm")
data = normalizer.transform(rescaledData)
mat = IndexedRowMatrix(
    data.select("num", "norm")\
        .rdd.map(lambda row: IndexedRow(row.num,
    row.norm.toArray()))).toBlockMatrix()
dot = mat.multiply(mat.transpose())
dot.toLocalMatrix().toArray()

dot_udf = psf.udf(lambda x,y: float(x.dot(y)),
    DoubleType())
data.alias("i").join(data.alias("j"), psf.col("i.
    num") < psf.col("j.num"))\
    .select(
        psf.col("i.num").alias("i"),
        psf.col("j.num").alias("j"),
        dot_udf("i.norm", "j.norm").alias("dot"))\
    .sort("i", "j")\
    .show()

tempcosine = data.alias("i").join(data.alias("j"),
    psf.col("i.num") < psf.col("j.num"))\
          .select(
              psf.col("i.num").alias("i"),
              psf.col("j.num").alias("j"),
              dot_udf("i.norm", "j.norm").alias("
    dot"))\
          .sort("i", "j")

sizeval = len(tempcosine.select("dot").collect())
valuesi = []
valuesj = []
valuepair = []

run = 0
while run < sizeval:
  valuesi.append(tempcosine.select("i").collect()[
    run][0])
  valuesj.append(tempcosine.select("j").collect()[
    run][0])
  valuepair.append(tempcosine.select("dot").collect
    ()[run][0])
  run = run + 1

#Se crea la matrix nxn, segun la cantidad de
    archivos
similmatrix = np.zeros((filecontent,filecontent))

pos = 0
#Se llena la triangular superior
while pos < sizeval:
  x = valuesi[pos]
  y = valuesj[pos]
  value = valuepair[pos]
  similmatrix[x-1][y-1] = value
  pos = pos + 1

i = 0
while i < filecontent:
  similmatrix[i][i] = 1
  i = i + 1

reverse = filecontent - 1
while  reverse >= 0:
  j = 0
  while j < reverse:
    similmatrix[reverse][j] =similmatrix[j][reverse
    ]
    j = j + 1
  reverse = reverse - 1

print (similmatrix)

SpectralClustering(2).fit_predict(similmatrix)

eigen_values, eigen_vectors = np.linalg.eigh(
    similmatrix)
orderedclusters = KMeans(n_clusters=2, init='k-
    means++').fit_predict(eigen_vectors[:, 2:
```

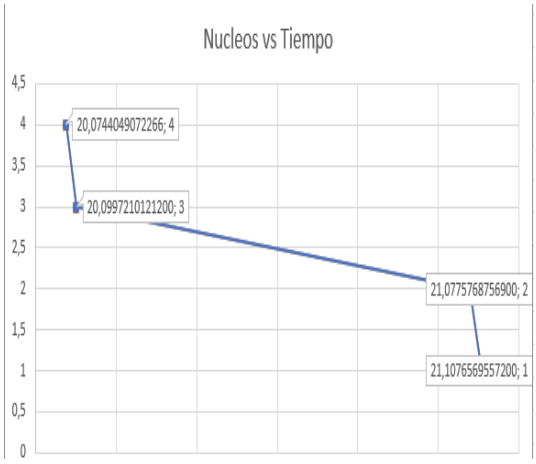
```
    filecontent])
print (orderedclusters)
kvalue = 2 #numero k
index = 0 #posicion que se esta analizando, significa que es el documento
group = 0 #grupos definidos segun el valor de k, comenzando en cero

recorrido = 0 #variable para iniciar el while
temp = "Cluster "
while recorrido < kvalue:
    temp += str(group) + ":"
    for cluster in orderedclusters:
        if cluster == group:
            temp += files[index]
            temp += ", "
        index += 1
    index = 0
    print (temp)
    temp = "Cluster "
    group += 1
    recorrido += 1

sc.stop()
```

V. Análisis de resultados (HPC vs Big Data) entre datasets e implementaciones

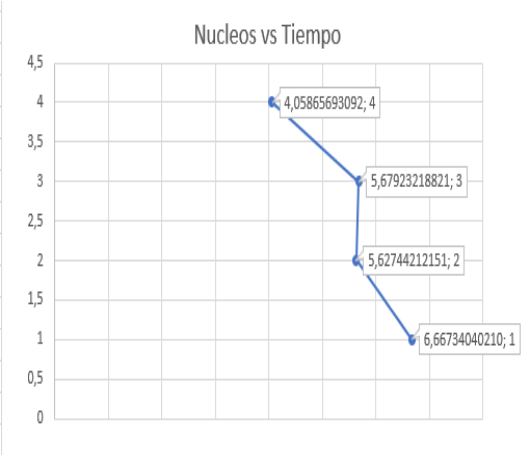
HPC

- 20 documentos del *dataset* de **Gutenberg**
- Un valor K igual a 10 (Gráfica 1)
- Un valor K igual a 4 (Gráfica 2)

Gráfica 1



Gráfica 2



Nota: Todos los valores de salida están expresados en segundos

Apache Spark

- 20 documentos del *dataset* de **Gutenberg** ejecución local
- Un valor K igual a 10
  Toma 1602.50241518 segundos procesar el dataset
- Un valor K igual a 4
  Toma 1609.29370499 segundos procesar el dataset

VI. Conclusiones

Mediante este proyecto se puede apreciar que la implementación realizada para computación de alto rendimiento es mejor que la implementación para apache spark (spark). Aunque para spark la implementación sea más fácil, dado que en este caso el motor de apache se encarga de paralelizar y utilizar los recursos como considere, se ve como el tiempo que toma procesar el dataset se ve seriamente castigado.

Es importante ver que cada tecnología tiene sus ventajas y desventajas, en HPC la complejidad y portabilidad representan un problema en el desarrollo de soluciones, pero en esta nos permite tener mas control de los recursos utilizados por lo que las soluciones aunque por lo general requieren de supercomputadores de pueden optimizar, en cambio tecnologías como apache spark que nos brindan una capa de abstracción adicional con el objetivo de facilitar, mejorar el desarrollo nos castiga en desempeño, aunque cuenta como herramientas que permiten mejorar y brindar soluciones más idóneas a los problemas.

Referencias

[1] Thomas. (2017). «Python: tf-idf-cosine: to find document similarity.» 2017, de Stack Overflow Sitio web: https://stackoverflow.com/questions/41504454/calculate-cosine-similarity-of-all-possible-text-pairs-retrieved-from-4-mysql-ta.

[2] Coursera. (2017). «Distance metrics: Cosine similarity.» 2017, de Coursera Sitio web: https://www.coursera.org/learn/ml-clustering-and-retrieval/lecture/yyegc/distance-metrics-cosine-similarity.

[3] «Set - Unordered collections of unique elements,» Python, 2017. [En línea]. Disponible en: https://docs.python.org/2/library/sets.html. [Último acceso: 2017].

[4] «Google's Python Class,» Google, [En línea]. Disponible en: https://developers.google.com/edu/python/. [Último acceso: 2017].

[5] «Similarity between two text documents,» stackoverflow, [En línea]. Disponible en: https://stackoverflow.com/questions/8897593/similarity-between-two-text-documents. [Último acceso: 2017].

[6] C. Perone, «Machine Learning :: Text feature extraction (tf-idf) – Part I,» 2011. [En línea]. Disponible en: http://blog.christianperone.com/2011/09/machine-learning-text-feature-extraction-tf-idf-part-i/.

[7] C. Perone, «Machine Learning :: Text feature extraction (tf-idf) – Part II,» 2011. [En línea]. Disponible en: http://blog.christianperone.com/2011/10/machine-learning-text-feature-extraction-tf-idf-part-ii/.

[8] C. Perone, «Machine Learning :: Cosine Similarity for Vector Space Models (Part III),» 2013. [En línea]. Disponible en: http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/.

[9] «Pairwise metrics, Affinities and Kernels,» scikit learn, [En línea]. Disponible en: http://scikit-learn.org/stable/modules/metrics.html#cosine-similarity.

[10] E. Fox, «k-means as coordinate descent,» University of Washington, [En línea]. Disponible en: https://www.coursera.org/learn/ml-clustering-and-retrieval/lecture/Fb58J/k-means-as-coordinate-descent.

[11] A. Trevino, «Introduction to K-means Clustering,» Data science, [En línea]. Disponible en: https://www.datascience.com/blog/k-means-clustering. [Último acceso: 2017].

[12] Calculating the cosine similarity between all the rows of a dataframe in pyspark. (s.f.). Obtenido de stackoverflow: https://stackoverflow.com/questions/46758768/calculating-the-cosine-similarity-between-all-the-rows-of-a-dataframe-in-pyspark

[13] Clustering - RDD-based API. (s.f.). Obtenido de Apache spark: https://spark.apache.org/docs/2.2.0/mllib-clustering.html

[14] Dey, S. (s.f.). Topic Clusters with TF-IDF Vectorization using Apache Spark. Obtenido de https://www.3pillarglobal.com/insights/topic-clusters-tf-idf-vectorization-using-apache-spark

[15] Extracting, transforming and selecting features. (s.f.). Obtenido de Apache spark: https://spark.apache.org/docs/2.2.0/ml-features.html#tf-idf

[16] Spark SQL, DataFrames and Datasets Guide. (s.f.). Obtenido de Apache spark: https://spark.apache.org/docs/latest/sql-programming-guide.html