文字列ビッグデータの関連度による可視化

石川 稔樹† 井上 潮‡

†‡東京電機大学工学部 〒120-8551 東京都足立区千住旭町 5

E-mail: †14ec010@ms.dendai.ac.jp, ‡inoueu@mail.dendai.ac.jp

あらまし 近年のデータの多様化・巨大化に伴い、構造化のなされていない大量のデータを扱う必要性が以前より増している。特に、Twitter や LINE などの SNS によって、大量の文字列データが生成され続けている。しかし、明確な解析手法が確立されていない、解析に時間がかかる、専門知識が必要になるなどの理由により、これらのデータを誰でもかんたんに解析できる状況にはなっていない。そこで、分散処理技術の Spark を使用して、Twitter の文字列データを関連度の観点から解析を行い、可視化するシステムを作成することで、文字列ビッグデータを誰でも比較的簡単に解析できる手法を検討した。

キーワード Apache Spark, Twitter, データ解析, 可視化, Word2Vec

1. はじめに

コンピュータ技術の発展とともに、様々な Web サービスが開発され、利用されてきた。Web サービスにおけるデータは、SNS や、ブログ等のテキストデータ、画像や、動画等のマルチメディアデータ等、その多くが構造化されていないデータである。特に、Twitter や Facebook などに代表される SNS においては、毎日大量の非構造化データが蓄積されている。これらのデータを解析することで、新たな価値を生み出すことができると考えられるため、非構造化データを解析したいというニーズが以前より増している。

しかし、これらの非構造化データを解析するための標準的な手法は確立されておらず、利用者が独自の解析手法で解析をする必要がある。また、蓄積された大量のデータを解析するのには膨大な時間がかかり、広範な専門知識が必要となる。

本論文では、文字列データを単語間の関連度の観点から解析し、可視化するシステムを提案する。単語間の関連度を評価するために、文章の形態素解析を行い、単語に分解したものを単語ベクトル表現に変換し、関連度を計算する。単語ベクトルを得るために、Word2Vec[1]を使用する。Word2Vec は、入力層、隠れ層、出力層の 3 層からなるニューラルネットワークで構成されており、形態素解析を行った単語の集合を入力することで、単語のベクトルを得ることができる。Word2Vec は、入力データのサイズが大きければ大きいほど処理に時間がかかる。そこで、Word2Vec の処理を分散処理フレームワークである Hadoop[2]と Spark[3]を使用して高速化を図る。

2. 関連研究

岡崎ら[4]は、Word2Vec において、データを複数のプロセッサあるいはコアで並列学習処理を行う際に速度が低下する原因が、キャッシュの偽共有という現象に起因するとし、並列処理を行っても偽共有が起きない仕組みを開発した。偽共有とは、並列処理をする際に、あるプロセッサのキャッシュラインが別のプロセッサのキャッシュライン上のメモリ内データを勝手に更新してしまうことである。

森田ら[5]は、Word2vec で得られた分散表現をもとに、単語の感情極性判定を行う手法を検討した。彼らは、「ある感情極性を持つ単語と類似度の高い単語は、同様の感情極性を持ちやすい」という仮説のもと、実験を行った。しかし、感情極性と分散表現上の近さには相関が無く、この仮説は棄却されている。

鹿島ら[6]は、Web 検索において、ユーザが与えるクエリが適切でない、曖昧であるために目的の情報が得られにくいという課題があることに対し、Word2Vec を使用した類語によるクエリの置換手法を提案した。単語ベクトル同士の平均を取ることで新たなベクトルを生み出し、そのベクトルを使用して類語を取得できるようにした。

大山ら[7]は、ゲームに関する情報を手に入れる手段として、Web でのキーワード検索では、不要な情報が多く含まれるという課題があり、その解決策として、ゲームのレビュー文を学習させた Word2Vec に、ユーザの要望を入力し、関連度が高いゲーム名を出力するシステムの開発を行った。

3. 研究目的と手法

本研究の目的は、Twitter の文字列データに含まれる単語の関連度を可視化することである。

関連度を計算するための手法として、ベクトルに変換した単語間のコサイン類似度を使用する。単語のベクトル化には Google が開発した Word2Vec を使用する。Word2Vec は、入力データサイズが大きいと処理に時間

がかかるため、複数台のコンピュータに処理を分散させる。そのために、Spark を核とした分散処理フレームワークを使用する。

以下に使用したフレームワークの概要を述べる。

3.1. Word2Vec

Word2Vec は、入力層、隠れ層、出力層の 3 層からなるニューラルネットワークで構成されている。今回使用するライブラリでは、Skip-gram という手法を用いて単語の推測を行う。Skip-gram は、指定されたウィンドウサイズ内で注目する単語と共起する単語の確率を求め、すべての単語に対する確率が最大となるようなベクトルを出力する手法である(図 1)。

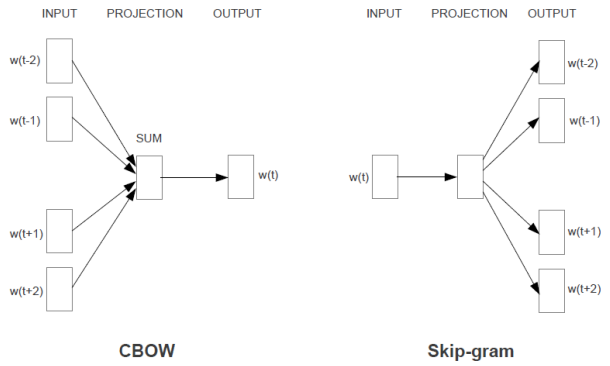

図 1.Skip-gram モデル

3.2. 分散処理フレームワーク

Hadoop は Apache Software Foundation によって開発されているオープンソースソフトウェアであり、分散処理のためのフレームワークである。HDFS(Hadoop Distributed File System)は、Hadoop の持つ機能の 1 つで、分散ファイルシステムを実現する。

HBase[8]は、Hadoop と同様に Apache によって開発されているオープンソースのキーバリューデータストアである。低レイテンシでアクセスすることができ、負荷に対して非常に強いという特性を持つ。HDFS 上に構築するため Hadoop との互換性も高く、Spark での分散処理を HBase のデータに対して行うことができる。また、HBase は、列指向型データベースであるために、他の KVS と違ってテーブルを柔軟に作成することができる。

Spark は、Hadoop や HBase と同様に、Apache によって開発されているオープンソースの分散処理フレームワークである。Hadoop の分散処理フレームワークに MapReduce があるが、ストレージへのアクセスが比較的頻回であるなどの欠点があり、場合によってはこれらがボトルネックとなることがある。Spark では、これらの欠点を補う構造となっており、MapReduce と比較したときに Spark の性能が何倍にも向上すると言われている。

4. システム構成

本研究で構築したシステムの概要を図 2 に示す。

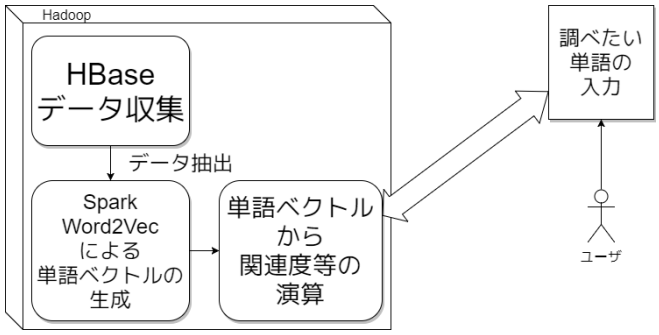

図 2.システム概要

Hadoop サーバ上では HBase が動作しており、Twitter のデータを逐一取得し、HBase に格納している。解析処理の際は、この HBase のデータを読み込む。

本システムではツイートデータの解析と解析によって得られた結果の表示という 2 つの処理を行う。ユーザはまず、ブラウザ上で解析に必要なパラメータを指定する。このパラメータは、デフォルトの値が指定してあり、デフォルト値のままで解析を行うこともできる。この入力を受け、Hadoop サーバで処理が開始される。以下に処理の流れについて記述する。

4.1.Mapper におけるデータの事前処理

はじめに、Word2Vec へデータを入力するための前準備を行う。元データとなるツイートには非常に多くのノイズが含まれており、そのまま学習を進めても、良い結果は得られない。そこで、Word2Vec に入力する前に事前処理を行う。

ツイートには独自の書き方がある。例えば、リツイートを指す RT、@から始まるユーザ ID やハッシュタグ#等がある。また、ツイートに URL が埋め込まれていることもある。これらは、形態素解析をする際のノイズにつながり、Word2Vec でベクトルにする際の確率に影響を及ぼす。そこで、これらの文字列を形態素解析する前に取り除く。取り除くものは、以下のとおりである。

- RT 時の RT@username
- リプライ時の@username
- ツイート中の URL
- 特殊記号
- ハッシュタグ記号#

次に、Word2Vec へ単語の集合を入力するため、読み込んだツイートに対して形態素解析器 MeCab[9]を用いて分かち書きを行う。ツイートには、ネットスラングや固有名詞、新語などが文章中に出現する場合が多くある。その為、MeCab 純正の辞書では、固有名詞の意図しない分解がなされてしまったり、新語に対応していなかったりと単語分解の精度が低下する。そこで、新たに辞書を追加することで対処する。追加する辞書は、MeCab-Ipadic-Neologd[10]である。この辞書は、オープンソースで、Web 上のデータを定期的に取得して更新されているため、新語や固有名詞などに強い特徴を持つ。

4.2.単語の可視化

分かち書きをしたツイートは、更にスペースで分解され、Word2Vec へ入力される。

図 3 に示すように、パラメータを指定することができ、このパラメータを受け取って Word2Vec の解析処理が開始される。

データが入力されたら学習が始まり、終了すると単語ベクトルモデルが生成される。図 4 に示すように、生成されたモデルデータから、検索フォームに入力したキーワードに対して、コサイン類似度を計算し、関連度が高い単語が樹形図に表示される。

表示された樹形図は、円をクリックすると更に下層の単語が表示されるようになっており、最大で 4 層まで表示されるようになっている。

図 3.解析パラメータの指定

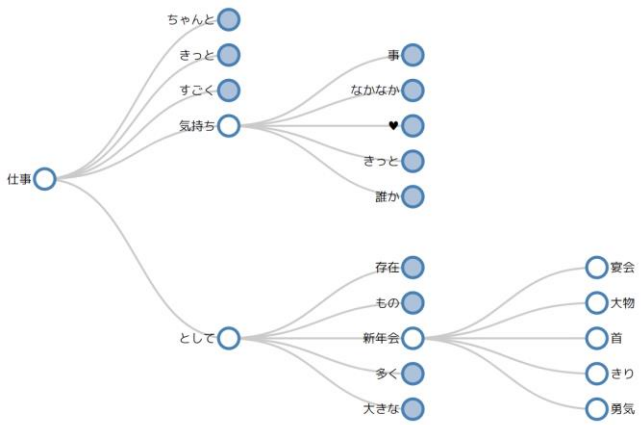

図 4.関連語の表示

5. 評価

5.1.評価方法

構築したシステムについて、以下の 2 つの観点から評価を行った。

(1) Spark による処理速度向上

本研究では、分散処理フレームワークを使用して、Word2Vec の処理を分散させている。この分散処理によってどの程度速度が向上しているかを測定し、評価する。

(2) Word2Vec における関連度の整合性

本研究では、Word2Vec を使用して文章を単語のベク

トルに変換し、コサイン類似度を使用して関連度を求めることで、テキストデータを可視化している。そこで、実際に関連度を求め、正誤率によって評価する。なお、正誤判定のモデルが無いので、正誤判定は主観での判断になる。

評価に際しては、表 1 に示すサーバ構成で、表 2 に示すデータを使用した。サーバは、マスタ 1 台、スレーブ 3 台の計 4 台で動作しており、HDFS は、240GB ×3 台で 720GB が割り当ててある。また、HBase には、ツイートを一意に識別できるように、ツイートに割り当てられているツイート ID と、ツイート本体を格納している。

表 1.Hadoop サーバ構成

OS	Linux CentOS7 64bit
CPU	Intel(R) Core(TM) i5-3470S CPU @ 2.90GHz 4Core 4Thread
メモリ	8.00GB
HDD	500GB
HDFS 容量(1 台あたり)	240GB

表 2.HBase テーブル

全データ件数	18521202 件 (9260601 件)
データ容量	5.46GB
カラム	ツイート ID ツイート本文

5.2. 評価結果

(1) Spark による処理速度向上

本研究では、Spark は Hadoop のクラスタマネージャ YARN 上に構築している。YARN では、コンテナ(エグゼキュータ)単位でリソースを管理しており、各コンテナにはコアとメモリを割り当てることができる。

したがって、いくつか条件を変え、処理速度を測定する。

測定は、5.1 で説明した環境のもと、コンテナの設定を変更して行う。使用するデータは、Twitter からサンプリング抽出し、HBase に格納したデータから、1 万ツイートを使用した。測定のタイミングは、HBase からデータを抽出し始めてから Word2Vec でベクトル化を行い、モデルデータをファイルとして出力するまでとした。

条件 1. スレーブサーバ数を変更

表 3 に示すリソースを割り当てたコンテナを 1 つ起動したスレーブサーバを 0 台から 3 台まで増やしていき、その台数によって速度が向上するかを検証する。

測定結果を図 5 に示す。

表 3.コンテナリソース割り当て

割当コア数	3
割当メモリ	6GB

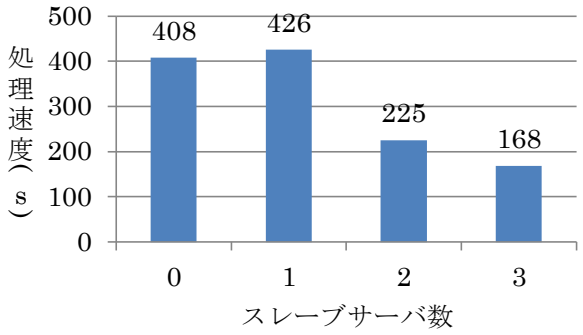

図 5.スレーブサーバ数の違いによる速度差

条件 2. 割当コア数を変更

割当コア数を 1 コアから 9 コアまで変え、その速度差を検証する。

表 4 に示すリソースを割り当てたコンテナを複数起動することで、コア数を変えて速度を測定する。

測定結果を図 6 に示す。

表 4.コンテナリソース割当

割当コア数	1
割当メモリ	1.5GB

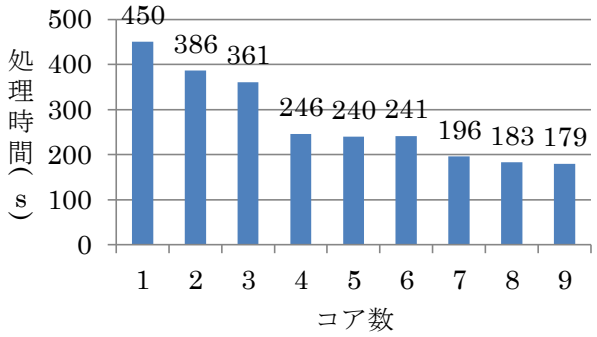

図 6.コア数の違いによる速度差

条件 3. HBase のリージョン数を変更

HBase に格納されているデータは、リージョンという単位で分割されている。Spark で処理する際のデータの取り出しは、各リージョンから均等にデータを取り出す事になっており、リージョンの数だけ、Spark での処理のタスクが発生する。

そこで、リージョン数の違いによって速度がどの程度異なるのか測定を行った。

リージョン数は、9、と 36 で測定を行った。

また、コンテナに割り当てたリソースは、表 4 のも

のとし、コア数が 3, 6, 9 になるようにコンテナを複数起動した。

測定結果を図 7 に示す。

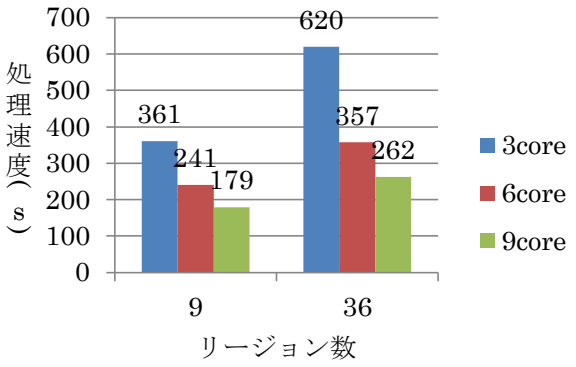

図 7.リージョン数の違いによる速度差

(2) Word2Vec における関連度の整合性

前提は(1)と同じとし､データ数も 100 万件抽出した。また、Word2Vec で処理をする際のパラメータとして、ウィンドウサイズを 5 とし、出現頻度が 2 以上の単語のみを解析の対象とした。この条件で 6 種類の注目単語に対して関連度が密な単語を上位から 10 単語ずつ求め、全 60 単語のうちの正誤率を求めた。

評価結果を表 4 に示す。関連の無い語句が多く見られ、正誤率が 31%と低い値になってしまった。

表 4.正誤率

正	誤	正誤率
19	41	31%

6. おわりに

本稿では、大量の文字列データを関連度という観点から分析し、可視化することができるシステムを提案し、その実装と評価について述べた。

本研究では、関連度によって可視化するために、コサイン類似度を使用した。コサイン類似度を求めるためには、単語をベクトル化する必要があり、そのために Word2Vec を使用した。入力データ量が大きくなると学習に時間がかかる問題を解決するため、Word2Vec の学習処理を分散処理フレームワーク上で実装した。評価結果のように、正しく分散され、スレーブ数が増えるほど速度が向上していることが実証された。また、コア数の違いについても検証を行い、速度が多少向上していることは実証されたが、大きな速度の向上には至らなかった。HBase のリージョン数についても検証を行った。リージョン数が増えるほど、処理も増えるので、全体の処理時間が長くなってしまうが、分散をすることで、速度を抑えることができる。また、出力された単語に関して、関連度があまり高くないものが多く見られた。これは、ノイズの削除や、パラメータの調整により改善できる可能性がある。

参 考 文 献

[1] Word2Vec
https://deeplearning4j.org/ja/word2vec

[2] Hadoop
https://hadoop.apache.org/

[3] Apache Spark
https://spark.apache.org/

[4] 岡崎直観、乾健太郎、“Word2Vec の並列実行時の学習速度の改善”、情報処理学会、Vol.2014-NL-217 No.8(2014)

[5] 森田晋也、白井靖人、“単語間類似度に基づいた単語感情極性の判定”、情報処理学会台 79 回全国大会 7Q-05

[6] 鹿島好央、北山大輔、“Word2Vec と Web 検索を用いた検索クエリ置換手法”、DEIM Forum 2017 C6-1

[7] 大山浩暉、竹川佳成、平田圭二、“レビュー文を考慮したゲーム推薦システムの実現に向けた単語の類似度調整の取り組み”、エンタテインメントコンピューティングシンポジウム(EC2017)2017 年 9 月

[8] HBase
https://hbase.apache.org/

[9] MeCab
http://taku910.github.io/mecab/

[10] MeCab-Ipadic-neologd
https://github.com/neologd/mecab-ipadic-neologd