令和元年度  
プロジェクトデザインⅢ プロジェクトレポート

Twitterを用いた各都道府県のトレンド  
の解析

金沢工業大学 工学部 情報工学科

4EP2-30 髙橋 史弥

指導教員 元木 光雄 准教授

令和２年1月21日提出

論文概要

流行を知ることは，人々が暮らしていくために必要不可欠である．そのため，テレビやインターネットを用いて，流行の調査を行う．しかし，それらの調査方法では，全国的な流行の調査は行えるが，地域ごとの流行を調査することが困難である．また，報道されるまでに時間が空いてしまうため最新の流行とは言えない．そこで，Twitterを用いることで，それらの問題が解決できると考えた．Twitterは，テレビやラジオなどで報道される前の話題や報道されないような小さな話題までリアルタイムで知ることが出来る．また，発信元のデータが含まれるツイートもあるため，地域ごとの調査が行えると考えた．本研究では，位置情報付きツイートを収集し，都道府県ごとの流行を知るための手法の開発を行う．

プロジェクト実施記録

|  |  |  |
| --- | --- | --- |
| 期間 | 作業内容 | 作業時間 |
| 4月から7月 | ツイートの取得，都道府県ごとの分類 | 112時間 |
| 8月から10月 | 共起ネットワークの生成 | 83時間 |
| 11月から12月上旬 | 特徴的なトレンドの抽出 | 110時間 |
| 12月下旬から1月 | 論文作成 | 60時間 |

目次

[第1章 序論 1](#_Toc31413590)

[第2章 準備 3](#_Toc31413591)

[2.1 グラフ理論の用語 3](#_Toc31413592)

[2.2 極大クリーク列挙 3](#_Toc31413593)

[2.3 データ研磨 4](#_Toc31413594)

[第3章 提案手法 6](#_Toc31413595)

[3.1 ツイートの取得と都道府県による分類 6](#_Toc31413596)

[3.2 共起ネットワークの生成 6](#_Toc31413597)

[3.3 都道府県ごとの特徴的なトレンド抽出 8](#_Toc31413598)

[第4章 実験による評価 9](#_Toc31413599)

[4.1 二値化とデータ研磨の閾値 9](#_Toc31413600)

[4.2 都道府県ごとの特徴的なトレンド抽出 11](#_Toc31413601)

[第5章 結論 13](#_Toc31413602)

[参考文献 14](#_Toc31413603)

[謝辞 15](#_Toc31413604)

[付録A ソースコード 16](#_Toc31413605)

[A.1 ツイートの取得と都道府県による分類 16](#_Toc31413606)

[A.1.1 search\_data.py 16](#_Toc31413607)

[A.1.2 sorting\_data.py 17](#_Toc31413608)

[A.2 共起ネットワークの生成 19](#_Toc31413609)

[A.2.1 Morphological\_analysis.py 19](#_Toc31413610)

[A.2.2 clique\_enumeration.py 20](#_Toc31413611)

[A.3 都道府県ごとの特徴的なトレンドの抽出 23](#_Toc31413612)

[A.3.1 create\_comparison 23](#_Toc31413613)

[A.3.2 comparison\_maximalClique 23](#_Toc31413614)

1. 序論

現在，スマートフォンやタブレット端末などの電子機器の急速な発展と普及により，若年層から高年層まで幅広い人がSNS (ソーシャルネットワーキングサービス)を利用している．日本国内におけるSNSの利用者は年々増加しており， 2018年末のSNS利用者数は7523万人(普及率75％)に達するとさている．さらに，SNSの利用者は，10代から20代の若年層に多い傾向があったが，SNS利用が当たり前となり，40から60年代以上の登録者数や利用者数も増加の傾向が見られる．このまま普及が進めば2020年末には，SNS利用者数が7937万人(普及率78.7％)に達するとも言われている[1]．

SNSの代表的なものとして，Facebook, Twitter, LINE, Instagramが挙げられる．これらのサービスの中で，Twitterは短い時間で自分の考えや気持ちをメッセージや画像，動画等を用いて，リアルタイムに伝えることが出来る．そのため， 様々な話題が集まるメディアとなっている．2019年4月時点でTwitterは，世界の月間アクティブアカウント数が3億3300万人，日本国内の月間アクティブアカウント数は4500万人となっている[2]．投稿されたツイートデータは国内だけでも膨大な量があるため，これを解析することで既存のメディアなどから得る事の出来なかったユーザーの率直な意見をリアルタイムに得る事が可能になった．この発信される話題は人々の暮らしや生活に大きく関わっている．しかし，Twitterは日本のトレンドしか公開していないため，身近な話題を知ることが出来ない．そこで，地域ごとでツイートを収集し解析することで，より自分自身に関係がある話題が入手できると考えた．

本研究では, 位置情報付きツイートに注目し, 都道府県という地域ごとでトレンドの解析を行う手法を提案する．この解析によって得られる情報は，多くのジャンルで役立つと考えられる．例えば，「兼六園　桜　綺麗」といった単語が抽出されたら観光ルートとしておすすめすることが出来るので観光推進に繋がる．また，「氷　滑る　危ない」といった単語が抽出されたら，事前に対策が出来るため事故の防止に繋がると考えた．

トレンドの抽出は，ある単語が含まれるツイート数や，同じツイートに含まれる単語同士の関係から抽出できると考える．また，特徴的なトレンドの抽出は，都道府県のツイートから抽出されるトレンドとそれ以外のツイートから抽出されるトレンドから同類のトレンドを見つけることで特徴的なトレンドが抽出できると考えた．

提案する方法として, 位置情報付きツイート取得するためにTwitterのstreaming APIを用いる. 次に，位置情報付きツイートを都道府県ごとに分類するために, YahooのYahoo!リバースジオコーダAPIとYahoo!ジオコーダAPIを用いる. さらに, 分類したデータごとにツイートに含まれる単語の共起関係を表すグラフを生成する. そして, 生成したグラフから極大クリーク列挙を行う.最後に, 都道府県の極大クリークとその都道府県以外のデータから生成した極大クリークを比較することで, 都道府県ごとの特徴的なトレンドを抽出できると考える.

本論文の構成は以下の通りである．第2章では，グラフ理論に関する用語の定義と，極大クリーク列挙，データ研磨についての説明を行う．第3章では，ツイートの取得と都道府県による分類，共起ネットークの生成，都道府県ごとの特徴的なトレンドの抽出の手法について述べる．第4章では，二値化とデータ研磨の閾値，都道府県ごとの特徴的なトレンド抽出を行った結果と考察を述べる.第5章では，結論を述べる．

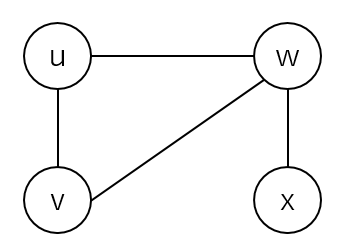
1. 準備

第2章では，グラフ理論に関する用語の定義と，極大クリーク列挙，データ研磨についての説明を行う．

* 1. グラフ理論の用語

グラフは組で表される．は有限集合で、はの二項関係である．は頂点集合(vertex set)と呼ばれ，その要素を頂点(vertex)と呼ぶ．は辺集合(edge set)と呼ばれ，その要素を辺(edge)と呼ぶ．また，グラフに対し，であり，かつの各要素の両端点が，に含まれるとき，で作られるグラフを部分グラフと呼ぶ．

において，頂点と辺で結ばれている頂点は頂点に隣接するといい，頂点に隣接する頂点集合を頂点の開近傍(open neighbor)と呼ぶ．頂点の開近傍はと表記する．である．の次数は，に隣接する頂点の数，つまりである．はのことであり閉近傍(closed neighbor)という．グラフの例を図1に示す．図1を参照するとは，の要素を持つ集合である．は近傍ではないため，には含まれない．

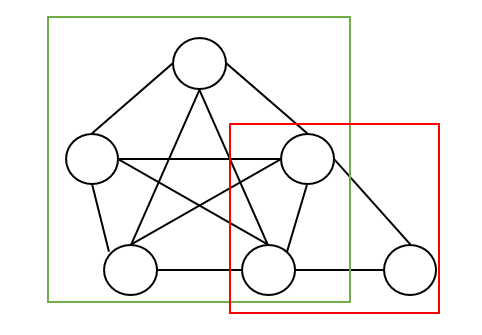


1. グラフの例

クリークとは全ての頂点間の組み合わせで，辺が繋がっている部分グラフのことである．そのクリークが他のクリークの部分グラフでなかった場合，このクリークは極大クリークとなる．

* 1. 極大クリーク列挙

似たものの頂点間に線を引いてグラフを作成する．同じグループに属するものは線で結ばれていて，全てのペアがお互いに結ばれているものがひとつのグループである．任意の頂点間に辺があるものをクリークと呼ぶ．極大クリークとは，これ以上頂点を増やすことが出来ないクリークのことである．この極大クリークをグラフの中から全て抽出することを極大クリーク列挙という．グラフに極大クリーク列挙を適応した例を図2に示す．図2では，四角で囲まれた部分が列挙される極大クリークとなる．



1. 極大クリークの例
   1. データ研磨

データ研磨[3]とは，与えられたグラフに対し，そのグラフにおける頂点や辺の構造を壊さずに与えられた条件によって明確なグラフに近づけるものである．具体的には，重み付けした頂点や辺に対し条件を課して，満たす頂点間には辺を張り，満たさない頂点間からは辺を除去する．これを複数回，または条件に対する閾値を変更することによって明確化されたグラフを得ることができる．データ研磨は微細な情報は失ってしまうが，ある程度以上の大きさを持つ構造を見えるようにするという効果がある．

データ研磨のアルゴリズムを図3に示す．研磨の方法は，頂点集合，辺集合と，研磨後の頂点集合，辺集合，指定した類似度下限値σを用意する．すべての頂点ペアに対して，類似度が指定した閾値よりも大きければ頂点ペアに辺を張り，そうでなければ辺を張らないというルールに従って新たにグラフを作成していく．新たにできたグラフを再度，同様の手法で研磨し，グラフ構造に変化がなくなるまで行う．

グラフ構造に変化がなくなるまで研磨することにより列挙される極大クリーク数が減り，より明確化される．そして列挙された極大クリークから，トレンドとなる単語を導き出せると考える．

|  |
| --- |
| Algorithm1 データ研磨アルゴリズム |
| **function** Polish  ：頂点集合，：辺集合，：類似度下限値  φ;φ  **for all** **do**  **for all** **do**  **if** npmi **then**        **end**  **end**  **end**  **return**  **end** |

1. データ研磨のアルゴリズム
2. 提案手法

第3章では，ツイートの取得と都道府県による分類，共起ネットークの生成，都道府県ごとの特徴的なトレンドの抽出の手法について述べる．

* 1. ツイートの取得と都道府県による分類

まず，TwitterのStreaming APIを利用して全国の位置情報付きツイートを収集する．Streaming APIの特徴は，リアルタイムのツイートが取得できることである．

次に，収集した位置情報付きツイートを都道府県ごとに分類する．そのために, Yahoo!リバースジオコーダAPIとYahoo!ジオコーダAPIを用いる．収集した位置情報付きツイートには，そのツイートがどこで発信されたか，緯度・経度または，バウンディングボックスという形で保存されている．緯度・経度で指定されている場合，それをYahoo!リバースジオコーダAPIに渡し，都道府県を一意に特定する．バウンディングボックスで指定されている場合，Yahoo!ジオコーダAPIに渡し，その範囲内の市町村を収集する，そして，その市町村が含まれる都道府県をツイートの発信元とする．本研究では，市町村データから複数の都道府県が導き出された場合，そのツイートは導き出された都道府県全てのデータとする．しかし，5つ以上の都道府県が導き出された場合，つぶやかれた範囲が広すぎると判断しそのツイートは除外する．

* 1. 共起ネットワークの生成

まず，都道府県ごとに分類したツイートに対し，MeCabによる形態素解析を行う．形態素解析とは，対象言語の文法や辞書と呼ばれる単語の品詞情報をもとに，文章を語に分け，それらの語の品詞を判別することをいう．また，品詞を指定して単語を取り出すことも可能である．表1に例文と形態素解析によって取り出される単語例を示す．本研究では，単語単体で意味のある名詞，動詞，形容詞， 形容動詞を抽出する．また，辞書は最新の単語の品詞情報が含まれるmecab-ipadic-neologdを利用する．

1. 例文と形態素解析によって取り出された単語群

|  |  |
| --- | --- |
| 例文 | 取り出された単語群 |
| サッカーの試合観戦に行く | サッカー，試合，観戦，行く |
| アーティストのライブが始まる | アーティスト，ライブ，始まる |

次に，形態素解析によって取り出された単語をもとに共起ネットワークを生成する．共起ネットワークとは，語と語の共起関係をネットワークにしたものである．単語を頂点とし，その単語が現れるツイート数を頂点に付与する．また，同一のツイートに現れる単語同士は辺で結び，共起回数を付与する．

そして,生成した共起ネットワークの各辺に対し，単語間の正規化自己相互情報量(NPMI: Normalized Pointwise Mutual Information)を計算する．NPMIの計算式を式(1)に示す．

式(1)のは単語の出現頻度を表す．具体的には単語を含むツイート数の全ツイート数に対する割合である．は単語が同時に出現する頻度を表している．NPMIは, 1.0から1.0の値をとり, 0から1.0になるにつれて，単語が同時に出現しやすい傾向にある．本研究では，NPMIに閾値を与え，NPMIの値が，あらかじめ指定した閾値よりも大きければそれ以上のNPMIの値を持つ辺のみを残し，それ以下の辺は削除する．この動作を各辺に対し，行うことで二値化を行う．二値化の閾値を大きく設定すれば，関係の強い頂点ペアのみを取り出すことができ，小さく設定すれば関係の弱い頂点ペアも取り出すことができる．そのため，閾値ごとに生成される共起ネットワークが異なる．よって，二値化を行う際は，最適な閾値を見つけ出す必要がある．

さらに，二値化を行った共起ネットワークに対し，極大クリーク列挙を行うと，類似した極大クリークが膨大になる．そのため，2.3節で述べたデータ研磨という手法を用いて，類似した極大クリークを減少させる必要がある. データ研磨は, 共起ネットワークのすべての頂点ペアについて，それぞれの頂点の近傍の類似度が，あらかじめ指定した閾値より大きければ辺を接続し，小さければ辺を削除する処理を行っている．二値化のときと同様に, 閾値ごとによって生成される共起ネットワークは異なるため，出力される辺数や極大クリーク数を考慮して，最適なデータ研磨の閾値を決める必要がある．

* 1. 都道府県ごとの特徴的なトレンド抽出

最後に, データ研磨を行った共起ネットークに対して, 極大クリーク列挙を行うことで，都道府県ごとのトレンド（単語集合）を抽出することができる．しかし，この段階では，全国的なトレンドの可能性もあるため，都道府県ごとの特徴的なトレンドとは言えない．そのため，調査する都道府県以外のツイートに対しても，同様の処理を行い, 極大クリーク列挙を行う．そして，その単語集合と都道府県の単語集合を比較し，都道府県側の単語集合で同じ単語が含まれる出現割合を計算する．その出現割合を求める計算式を式(2)に示す。

式(2)のは都道府県のトレンド，は全国なトレンドを示している．具体的には，はトレンドに含まれる単語の総数，は互いのトレンドに含まれる同一の単語の総数を示している．本研究では，その出現割合が80％を超えた場合，同じトレンドとみなし，そのトレンドは削除する．以上により，都道府県ごとの特徴的なトレンドが抽出できる．

1. 実験による評価

第4章では，二値化とデータ研磨の閾値，都道府県ごとの特徴的なトレンド抽出を行った結果と考察を述べる.

* 1. 二値化とデータ研磨の閾値

本実験では，2019年11月30日20時から2019年11月30日22時の間につぶやかれた位置情報付きツイートを実験対象とした．収集したツイート数は32,939件である．これを都道府県ごとに分類すると，1つのツイートが複数の都道府県のツイートに含まれることがあるため，32,972‬件と合計ツイート数が増大した．その中で，最もツイート数が多かった「東京都」のトレンドの抽出を実験対象とした．表2に，都道府県ごとのツイート数を示す．

1. 都道府県ごとのツイート数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 都道府県 | ツイート数 | 都道府県 | ツイート数 | 都道府県 | ツイート数 |
| 北海道 | 1,189 | 石川県 | 297 | 岡山県 | 481 |
| 青森県 | 261 | 福井県 | 170 | 広島県 | 578 |
| 岩手県 | 224 | 山梨県 | 251 | 山口県 | 333 |
| 宮城県 | 548 | 長野県 | 374 | 徳島県 | 106 |
| 秋田県 | 195 | 岐阜県 | 477 | 香川県 | 274 |
| 山形県 | 232 | 静岡県 | 1,017 | 愛媛県 | 287 |
| 福島県 | 393 | 愛知県 | 2,047 | 高知県 | 120 |
| 茨城県 | 816 | 三重県 | 413 | 福岡県 | 1,147 |
| 栃木県 | 342 | 滋賀県 | 344 | 佐賀県 | 270 |
| 群馬県 | 644 | 京都府 | 736 | 長崎県 | 221 |
| 埼玉県 | 1,661 | 大阪府 | 2,294 | 熊本県 | 206 |
| 千葉県 | 1,627 | 兵庫県 | 1,137 | 大分県 | 160 |
| 東京都 | 6,429 | 奈良県 | 273 | 宮崎県 | 183 |
| 神奈川県 | 2,604 | 和歌山県 | 175 | 鹿児島県 | 200 |
| 新潟県 | 448 | 鳥取県 | 136 | 沖縄県 | 311 |
| 富山県 | 190 | 島根県 | 151 |  |  |

まず，東京都のツイートから生成した共起ネットワークの二値化を行った．二値化の閾値は，0.1から0.9の0.1刻みごとに設定している．表3に二値化の閾値ごとの辺数を示す．

1. 二値化の閾値ごとの辺数

|  |  |
| --- | --- |
| 二値化の閾値 | 辺数 |
| 0.1 | 77,243 |
| 0.2 | 62,659 |
| 0.3 | 45,809 |
| 0.4 | 30,274 |
| 0.5 | 17,203 |
| 0.6 | 6,425 |
| 0.7 | 1,486 |
| 0.8 | 881 |
| 0.9 | 401 |

表3より，閾値の値を大きくすると辺数が減少し，関係性の強い単語同士の辺が残っていることが分かる．今回は，単語同士の関係性が高い0.6から0.9の範囲から，最も辺数が減少したことで関係性の強い辺だけが残った0.6を二値化の閾値と決定した．

次に，二値化された共起ネットワークにデータ研磨を行った．データ研磨の閾値も0.1刻みとし，閾値が小さすぎるとグラフが膨大な辺数になってしまうため, 0.6から0.9までを閾値の範囲とした．表4にデータ研磨の閾値ごとの辺数，極大クリーク数を示す．

1. データ研磨の閾値ごとの辺数，極大クリーク数

|  |  |  |
| --- | --- | --- |
| データ研磨の閾値 | 辺数 | 極大クリーク数 |
| 0.6 | 11,648 | 330 |
| 0.7 | 5,419 | 405 |
| 0.8 | 2,619 | 189 |
| 0.9 | 1,199 | 35 |

表4より，0.7の閾値の時に辺数が大きく減少していることが分かる．さらに，クリークの数も多いため，多くのトレンドを抽出できると考えた． そのため，0.7をデータ研磨の閾値と決定した．

* 1. 都道府県ごとの特徴的なトレンド抽出

二値化の閾値を0.6，データ研磨の閾値を0.7に指定し「東京都」の抽出した極大クリークの例を表5にいくつか示す．

1. 「東京都」のツイートから抽出した極大クリークの例

|  |
| --- |
| 鏡, 目白, ライトアップ, X, モミジ, 庭園, 照らす, 秋 |
| 表参道, (), イルミネーション, よる, 並ぶ |
| 20時, 連, 間, 引ける, ミッション, 分, シノアリス, 配布, チケット, リニューアル, 本日, 上位, 日, 4位, 22時, トレンド |
| U1, 小野路, ゼルビア, 町田ゼルビア, 相模原, メトロポリタンリーグ |

表5の単語群を見ると，「秋の庭園ライトアップ」や「表参道イルミネーション2019」，アプリのイベントやサッカーの試合と思われる単語群が抽出された．それらの単語群について調べた結果，11月30日の前後で開催されたイベントや行われる前のイベントであることが確認できたため，トレンドを抽出できたと言える．

次に，東京都の特徴的なトレンドを抽出するために，東京都以外のツイートに対しても，同様の処理を行い，極大クリーク列挙を行う．表5に東京都以外の抽出した極大クリークの例を表6に示す．

1. 東京都以外のツイートから抽出した極大クリークの例

|  |
| --- |
| 横浜, 散歩, みなとみらい, 夜景, 海, 休日, 空, Yo, キレイ,イルミネーション |
| 初, 田舎, 倍率, 嵐, ライブビューイング, もと, 低い |
| トレンド, 4位, リニューアル, チケット, 間, 配布, シノアリス, 20時, 引ける, 分, ミッション, 日, 本日, 連, 22時, 上位 |
| 炎の体育会TV, 困り顔, 体育会, 炎, 対抗, 松本人志, チャレンジ" |

表6の単語群を見ると，「横浜イルミネーション」や「嵐ライブビューイング」，アプリのイベントやテレビ番組と思われる単語群が抽出された．それらの単語群についても調べた結果，11月30日の前後で開催されたイベントや行われる前のイベントであることが確認できたため，トレンドを抽出できたと言える．

最後に，「東京都」の抽出した極大クリーク405件と東京都以外の抽出した極大クリーク1,803件を比較する．比較した結果，「東京都」のトレンド405件の内，10件が全国的なトレンドとして除外された．除外された極大クリークの比較結果をいくつか表7に示す．

1. 除外された極大クリークの比較結果

|  |  |  |
| --- | --- | --- |
| 極大クリーク(東京都) | 極大クリーク(東京以外) | 単語の出現割合(％) |
| 20時, 連, 間, 引ける, ミッション, 分, シノアリス, 配布, チケット, リニューアル, 本日, 上位, 日, 4位, 22時, トレンド | 20時, 連, 間, 引ける, ミッション, 分, シノアリス, 配布, チケット, リニューアル, 本日, 上位, 日, 4位, 22時, トレンド | 100 |
| クリア, セット, 有馬, 馬券, スペシャル, 黄金,チャンス, 64枚, ステップ, 盗む, 有馬記念, 企画, Lupin, ルパン三世, 電子マネー, ギフト, 1万円, シ, 五, 当たる, | クリア, セット, 有馬, 馬券,スペシャル, 黄金, チャンス,  64枚, ステップ, 盗む, 有馬記念, 企画, Lupin, ルパン三世,  電子マネー, ギフト, 1万円,  うま, コンテンツ, すべて, | 85 |
| 勝つ, 残り, お祭り, わけ, アジア, 劇, 優勝, 決める, 試合, 間違い, 4点差, 考える, | 勝つ, 残り, お祭り, わけ, アジア, 劇, 優勝, 決める, 試合, 間違い, 4点差, 勝ち点, 1試合, ボンボン | 91 |

以上により，「東京都」のトレンドから全国的なトレンドが除外され，東京都の特徴的なトレンドだけを抽出することが出来たと考えられる.

1. 結論

本研究では，取得したツイートに対し形態素解析と共起ネットワークの生成，データ研磨，極大クリーク列挙を行ったことで，都道府県ごとのトレンドを抽出することが抽出することができた．さらに，調査する都道府県のツイートとそれ以外のツイートから抽出したデータを比較することで，都道府県ごとの特徴的なトレンドを抽出することができた．今後の課題として，全国のトレンドを位置情報付きツイートに限定せず，全てのツイートから抽出する．

参考文献

1. ICT総研，2018年度 SNS利用動向に関する調査, https://ictr.co.jp/report/20181218.html，2019年12月16日参照.
2. [We Love Social](https://blog.comnico.jp/we-love-social/)，【2019年11月更新】人気SNSの国内＆世界のユーザー数まとめ（Facebook、Twitter、Instagram、LINE）, https://blog.comnico.jp/we-love-social，2019年12月16日参照.
3. 宇野他, ”データ研磨によるクリーク列挙クラスタリング”, 情報処理学会研究告，Vol. 2014-AL-146，No. 2，pp. 1-8, 2014.

謝辞

本研究にあたり，ご指導をいただいた元木光雄准教授に深く感謝いたします．また，元木研究室の皆様にも様々なご意見をいただき感謝いたします．

ソースコード

ツイートの取得と都道府県による分類

search\_data.py

日本の位置情報付きツイートを収集するプログラムを図4に示す．図4のget\_tweets関数は，TwitterのstreamingAPIを用いて位置情報付きツイートの収集を行う．write\_file関数は，収集したツイートをjson形式で保存する．

|  |
| --- |
| from requests\_oauthlib import OAuth1Session  import json  import datetime  import os  import sys  # ツイートを取得する  def get\_tweets():  api\_key = ''  api\_secret\_key = ''  access\_token = ''  access\_token\_secret = ''  # streamingAPIのURL  url = 'https://stream.twitter.com/1.1/statuses/filter.json'  # 認証  Twitter=OAuth1Session(api\_key,api\_secret\_key,access\_token, access\_token\_secret)  # 全都道府県の指定  params = {'locations': '139.317000,41.356001,149.080000,45.802000,'  '132.194001,32.642000,143.005000,41.356000,'  '128.562000,30.200001,132.194000,35.000000,'  '123.367000,23.840000,130.485000,30.200000'}  req = twitter.post(url, data=params, stream=True)  last\_time = datetime.datetime.now() + datetime.timedelta(hours=2) # 終了時間  tweets = []  if req.status\_code == 200:  for tweet in req.iter\_lines():  now\_time = datetime.datetime.now() # 時間の更新  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  if now\_time > last\_time:  break  try:  tweets.append(json.loads(tweet.decode("utf-8")))  except:  pass  return tweets  else:  print("Error: %d" % req.status\_code)  sys.exit()  # 取得したツイートの書き込み  def write\_file(tweets,create\_folder,create\_file):  os.makedirs(create\_folder, exist\_ok=True)  with open(os.path.join(create\_folder,create\_file),'w',encoding='utf-8') as file:  json.dump(tweets,file , ensure\_ascii=False, indent=2)  if \_\_name\_\_ == '\_\_main\_\_':  create\_folder = '2019-11-30-20'  create\_file = 'tweets.json'  tweets = get\_tweets()  write\_file(tweets,create\_folder,create\_file) |

1. 日本の位置情報付きツイートを収集するプログラム

sorting\_data.py

位置情報付きツイートを都道府県ごとに分類するプログラムを図5に示す．図5のprefectureCode1\_limit関数とprefectureCode2\_limit関数は，YahooAPIの利用上限が最大に達した時の処理を行う．get\_prefectureCode1関数とget\_prefectureCode2関数は，YahooAPIを用いて都道府県の特定を行う．sort\_tweets関数は特定した都道府県を用いて，ツイートの分類を行う．write\_file関数は，分類したツイートをjson形式で都道府県ごとに保存する

|  |
| --- |
| import urllib.parse  import urllib.request  import json  import os  import time  appid = ''  # YahooAPIの利用上限の回避  def prefectureCode1\_limit(lat,lon):  while True:  url = 'https://map.yahooapis.jp/geoapi/V1/reverseGeoCoder?'  params = urllib.parse.urlencode({'appid': appid, 'lat': lat , 'lon': lon , 'output': 'json'})  response = urllib.request.urlopen(url + params)  if response.status == 200:  break  else:  print('上限に達しております')  time.sleep(21600)  return response    # YahooAPIの利用上限の回避  def prefectureCode2\_limit(bbox,page):  while True:  url = 'https://map.yahooapis.jp/geocode/V1/geoCoder?'  params = urllib.parse.urlencode({'appid': appid, 'bbox': bbox, "al": 2, 'results': 100, 'output': 'json','page': page})  response = urllib.request.urlopen(url + params)  if response.status == 200:  break  else:  print('上限に達しております')  time.sleep(21600)  return response  # 緯度経度から県コード取得  def get\_prefectureCode1(lat,lon):  try:  url = 'https://map.yahooapis.jp/geoapi/V1/reverseGeoCoder?'  params = urllib.parse.urlencode({'appid': appid, 'lat': lat , 'lon': lon , 'output': 'json'})  response = urllib.request.urlopen(url + params)  if response.status == 403:  response = prefectureCode1\_limit(lat, lon)  data = json.load(response)  return data['Feature'][0]['Property']['AddressElement'][0]['Code']  except:  print("正しいデータではありません")  # bounding\_boxから県コードを予測し取得  def get\_prefectureCode2(bbox):  try:  url = 'https://map.yahooapis.jp/geocode/V1/geoCoder?'  params = urllib.parse.urlencode({'appid': appid, 'bbox': bbox, "al": 2, 'results': 100, 'output': 'json','page': 1})  response = urllib.request.urlopen(url + params)  if response.status == 403:  response = prefectureCode2\_limit(bbox, 1)  data = json.load(response)  if data['ResultInfo']['Total'] > 200:  return None  else:  if data['ResultInfo']['Total'] > 100:  params = urllib.parse.urlencode({'appid': appid, 'bbox': bbox, "al": 2, 'results': 100, 'output': 'json', 'page': 2})  response = urllib.request.urlopen(url + params)  if response.status == 403:  response = prefectureCode2\_limit(bbox, 2)  data.update(json.load(response)['ResultInfo'])  prefectureCodes = []  for geo in data['Feature']:  # 市町村コードなので上2桁を取得し、都道府県を導き出す。  prefectureCodes.append(geo['Property']['GovernmentCode'][0:2])  return set(prefectureCodes)  except:  print("正しいデータではありません")  # 都道府県の分類  def sort\_tweets(use\_folder,use\_file):  with open(os.path.join(use\_folder,use\_file),'r', encoding="utf-8") as file:  tweets = json.load(file)  prefectureTweetData = [[] for i in range(48)]  for t in tweets:  if t['geo'] != None:  lat, lon = t['geo']['coordinates']  prefectureCode = get\_prefectureCode1(lat, lon)  if prefectureCode != None:  prefectureTweetData[int(prefectureCode)].append(t)  else:  left\_lat,left\_lon= t['place']['bounding\_box']['coordinates'][0][0]  right\_lat,right\_lon = t['place']['bounding\_box']['coordinates'][0][2]  if left\_lat == right\_lat or left\_lon == right\_lon: # bboxの緯度経度が同じ  pass  else:  bbox = str(left\_lat) + ',' + str(left\_lon) + ',' + str(right\_lat) + ',' + str(right\_lon)  prefectureCodes = get\_prefectureCode2(bbox)  if prefectureCodes != None and len(prefectureCodes) < 5: # 都道府県が4以下  for prefectureCode in prefectureCodes:  prefectureTweetData[int(prefectureCode)].append(t)  return prefectureTweetData  # 分類したツイートを都道府県ごとに書き込む  def write\_file(prefectureTweetData,use\_folder):  for p, tweets in enumerate(prefectureTweetData):  with open(os.path.join(use\_folder, 'tweets' + str(p) + '.json'), 'w', encoding='utf-8') as file:  json.dump(tweets, file, ensure\_ascii=False, indent=2)  if \_\_name\_\_ == '\_\_main\_\_':  use\_folder = '2019-11-30-20'  use\_file = 'tweets.json'  prefectureTweetData = sort\_tweets(use\_folder,use\_file)  write\_file(prefectureTweetData, use\_folder) |

1. 位置情報付きツイートを都道府県ごとに分類するプログラム

共起ネットワークの生成

morphological\_analysis.py

形態素解析を行うプログラムを図6に示す．図６のremove\_url関数は，ツイートに含まれるURLの削除を行う．morphological\_analysis関数は，形態素解析を行い，ツイートから名詞，動詞，形容詞，形容動詞の抽出を行う．

|  |
| --- |
| import re  import json  import os  import MeCab  import csv  # 不要な文字列の消去  def remove\_url(text):  text = re.sub(r'https?://[\w/:%#\$&\?\(\)~\.=\+\-]+', "", text) # URLの消去  return text  # 形態素解析  def morphological\_analysis(text):  part\_of\_speech = {'名詞', '動詞', '形容詞', '形容動詞'}  words = set() # 単語の重複を防ぐ  m = MeCab.Tagger('-d /usr/lib/x86\_64-linux-gnu/mecab/dic/mecab-ipadic-neologd') # 辞書を指定  # surface(単語)feature(品詞情報)を持つ解析結果を代入  node = m.parseToNode(text)  while node:  feature = node.feature.split(',')  if feature[0] in part\_of\_speech:  words.add(feature[6]) # 原形保存  node = node.next  return words  if \_\_name\_\_ == '\_\_main\_\_':  use\_folder = '2019-11-30-20/trendo\_in\_13'  use\_file = 'tweets\_all.json'  create\_file = 'tweets\_all\_word\_list.csv'  words\_list = []  # jsonファイルからツイートデータを呼び出す。  with open(os.path.join(use\_folder, use\_file), 'r', encoding="utf-8") as file:  tweets = json.load(file)  # 保存したツイートデータのテキストを形態素解析。  for t in tweets:  tweet\_text = remove\_url(t['text'])  text\_words = morphological\_analysis(tweet\_text)  words\_list.append(text\_words)  # 形態素解析を行った文字のリストをcsvファイルとして保存。  with open(os.path.join(use\_folder, create\_file), 'w', encoding='utf-8') as file:  writer = csv.writer(file,lineterminator='\n')  writer.writerows(words\_list) |

1. 形態素解析を行うプログラム

clique\_enumeration.py

二値化とデータ研磨，極大クリーク列挙を行うプログラムを図7に示す．図7のadd\_tweet\_to\_graph関数は共起ネットワーク生成を行う．remove\_node関数は，不要な頂点の削除を行う．npmi関数はnpmiの計算を行う．binarization\_data\_polish関数は，二値化とデータ研磨と極大クリーク列挙を行い，極大クリークを抽出し，json形式で保存する．

|  |
| --- |
| import json  import csv  import networkx as nx  from itertools import combinations  import matplotlib.pyplot as plt  import datapolish as dp  import os  # 共起ネットークの生成  def add\_tweet\_to\_graph(words,g):  for w in words:  if w in g: # 単語が頂点集合に存在するか  g.node[w]['weight'] += 1 # 頂点の属性+1  else:  g.add\_node(w, weight=1) # 頂点の追加  for u, v in combinations(words, 2):  if g.has\_edge(u, v): # 指定した辺が存在するか  g[u][v]['weight'] += 1 # 辺の属性+1  else:  g.add\_edge(u, v, weight=1) # 辺の追加  return words  # 不要なnodeの消去  def remove\_node(g):  for n in list(g.nodes):  if g.node[n]['weight'] < 4: #出現回数3回以下を削除  g.remove\_node(n)  def npmi(g):  for u, v in g.edges():  g[u][v]['npmi'] = dp.npmi(g[u][v]['weight'] / len(words\_list), g.node[u]['weight'] / len(words\_list),  g.node[v]['weight'] / len(words\_list))  # 二値化の閾値データ確認  def threshold\_nichika(g):  count\_Threshold = [0.5,0.6, 0.7, 0.8, 0.9]  for i in count\_Threshold:  B = dp.graph.binarize\_graph(g, 'npmi', i)  print(i)  print('二値化の 辺数 = ', len(B.edges))  print('二値化の 極大クリーク数 = ', len(list(nx.find\_cliques(B))))  # 研磨の閾値データ確認  def threshold\_kenma(B):  count\_Threshold = [0.8]  for j in count\_Threshold:  C = dp.iterate\_data\_polish(B, dp.sim\_npmi, j, 100, verbose=False)  cliquesC = []  lsolated\_vertex = []  for i in nx.find\_cliques(C):  if len(i) > 2: # 孤立頂点以外を残す  cliquesC.append(i)  else:  lsolated\_vertex.append(i)  print(j)  print('データ研磨の 辺数 = ', len(C.edges))  print('データ研磨の 極大クリーク数 = ', len(cliquesC))  print('データ研磨の 孤立頂点数 = ', len(lsolated\_vertex))  # 極大クリークの生成  def binarization\_data\_polish(g,folder):  #threshold\_nichika(g)  B = dp.graph.binarize\_graph(g, 'npmi', 0.6)  print('二値化の 頂点数 = ', len(B.nodes))  print('二値化の 辺数 = ', len(B.edges))  print('二値化の 極大クリーク数 = ', len(list(nx.find\_cliques(B))))  #threshold\_kenma(B)  C = dp.iterate\_data\_polish(B, dp.sim\_npmi, 0.7, 100, verbose=False)  cliquesC = []  lsolated\_vertex = []  for i in nx.find\_cliques(C):  if len(i) > 2: # 孤立頂点以外を残す  cliquesC.append(i)  else:  lsolated\_vertex.append(i)  print('データ研磨の 頂点数 = ', len(C.nodes))  print('データ研磨の 辺数 = ', len(C.edges))  print('データ研磨の 極大クリーク数 = ', len(cliquesC))  print('データ研磨の 孤立頂点数 = ', len(lsolated\_vertex))  tatal = 0  for k in cliquesC:  for v in k:  tatal += G.node[v]['weight']  sortedDict = sorted(cliquesC, key=lambda k: tatal / len(k))  # folderの生成  os.makedirs(folder, exist\_ok=True)  # 極大クリークの単語リストをjsonに保存  with open(os.path.join(folder,'cliques\_all.json'), 'w', encoding='utf-8') as file:  json.dump(sortedDict, file, ensure\_ascii=False, indent=1, sort\_keys=False, separators=(',', ': '))  # グラフの表示  def view\_graph(g,folder):  plt.figure(figsize=(6, 6)) # figure(図)の縦横の大きさを設定  pos = nx.spring\_layout(g) #座標の取得  nx.draw\_networkx\_nodes(g, pos, node\_size = 5) # nodeの可視化  nx.draw\_networkx\_edges(g, pos, width = 0.5) # edgeの可視化  plt.axis("off") # 座標軸をoff  plt.savefig(os.path.join(folder,'cliques13.png')) # 画像として保存  if \_\_name\_\_ == '\_\_main\_\_':  use\_folder = "2019-11-30-20/trendo\_in\_13"  use\_file = "tweets\_all\_word\_list.csv"  G = nx.Graph()  with open(os.path.join(use\_folder, use\_file), 'r', encoding='utf-8') as file:  reader = csv.reader(file)  words\_list = [row for row in reader]  for words in words\_list:  add\_tweet\_to\_graph(words, G)  remove\_node(G)  npmi(G)  binarization\_data\_polish(G,use\_folder)  view\_graph(G,use\_folder) |

1. 二値化とデータ研磨，極大クリーク列挙を行うプログラム

都道府県ごとの特徴的なトレンドの抽出

create\_comparison

対象とする都道府県以外のツイートの収集を行うプログラムを図8に示す．図8のread\_file関数は，トレンド抽出対象の都道府県以外からツイートの収集を行う．write\_file関数は，トレンド抽出対象の都道府県以外のツイートをjson形式で保存する．

|  |
| --- |
| import os  import json  import shutil  # 対象となる都道府県以外のデータを保存  def read\_file(use\_folder, prefectureCode):  other\_tweets = []  for t in range(48):  if t != prefectureCode:  with open(os.path.join(use\_folder, 'tweets' + str(t) + '.json'), 'r', encoding='utf-8') as file:  tweets = json.load(file)  for t in tweets:  other\_tweets.append(t)  return other\_tweets  # 対象となる都道府県以外のデータを書き込む  def write\_file(use\_folder, other\_tweets):  os.makedirs(use\_folder + "/trendo\_in\_" + str(prefectureCode), exist\_ok=True)  with open(os.path.join(use\_folder + "/trendo\_in\_" + str(prefectureCode), 'other\_tweets.json'), 'w',  encoding='utf-8') as file:  json.dump(other\_tweets, file, ensure\_ascii=False, indent=2)  if \_\_name\_\_ == '\_\_main\_\_':  use\_folder = '2019-11-30-20'  # 調べたい都道府県の県コード  prefectureCode = 13  other\_tweets = read\_file(use\_folder, prefectureCode)  write\_file(use\_folder, other\_tweets)  shutil.copy(use\_folder + '/tweets' + str(prefectureCode), use\_folder + "/trendo\_in\_" + str(prefectureCode)) |

1. 対象とする都道府県以外のツイートの収集を行うプログラム

comparison\_maximalClique

都道府県の特徴的なトレンドを抽出するプログラムを図9に示す．図9のread\_file関数は，都道府県の極大クリークと全国の極大クリークの読み込みを行う．common\_delete関数では, 都道府県の極大クリークと全国の極大クリークを比較し，都道府県側の単語集合で同じ単語が含まれる出現割合を計算し，削除する極大クリークを抽出する．write\_file関数は，残った極大クリークをjson形式で保存する．

|  |
| --- |
| import os  import json  def read\_file(use\_folder, use\_file):  with open(os.path.join(use\_folder, use\_file), 'r', encoding='utf-8') as file:  clique = json.load(file)  return clique  def common\_delete(main\_clique,sub\_clique):  difference = list(set(main\_clique) - set(sub\_clique))  probability = 1 - (len(difference) / len(main\_clique))  if probability > 0.8:  return main\_clique  def write\_file(new\_main\_clique,use\_folder):  with open(os.path.join(use\_folder,"new\_clique.json"),'w',encoding='utf-8') as file:  json.dump(new\_main\_clique,file , ensure\_ascii=False, indent=2)  if \_\_name\_\_ == '\_\_main\_\_':  use\_folder = '2019-11-30-20/trendo\_in\_13'  # 調べる対象の都道府県のクリーク  main\_cliqueFile = 'cliques13.json'  # 調べる対象の都道府県以外のクリーク  other\_cliqueFile = 'cliques\_all.json'  main\_cliques = read\_file(use\_folder, main\_cliqueFile)  other\_cliques = read\_file(use\_folder, other\_cliqueFile)  delete\_cliques = []  for main\_clique in main\_cliques:  for other\_clique in other\_cliques:  clique = common\_delete(main\_clique, other\_clique)  if clique != None:  delete\_cliques.append(clique)  # 削除するクリークの重複を削除  delete\_cliques = list(map(list, set(map(tuple, delete\_cliques))))  # 指定したクリークの削除  for delete\_clique in delete\_cliques:  main\_cliques.remove(delete\_clique)  write\_file(main\_cliques, use\_folder) |

1. 都道府県の特徴的なトレンドを抽出するプログラム