

進捗報告

氷見夏輝

富山県立大学
u220051@st.pu-toyama.ac.jp

December 26, 2025

平井さんのあげていた課題

2/1

今後の課題

実行時間の短縮

- マルチプロセスや分散処理を用いたスクレイピング
- 近年開発された Vibrato のような高速な分かち書き手法

外れ値に強いクラスタリング手法の検討

- 外れ値が大きいと、クラスタ数の決定がうまくいかず大雑把な分類になってしまふ。そのため、k-means 以外の重心を用いるクラスタリング手法を検討する必要がある。

3D グラフに表示されている単語の表示方法

- 現在はすべての単語に対して一定の大きさと色で表示しているが、単語の出現頻度などによって、大きさや色を変えることで、より効率的な探索が行えると考える。

図 1: 画像 A

課題

実行時間が長すぎる。

外れ値が大きいと、クラスタ数の決定がうまくいかず大雑把な分類になってしまう。そのため、k-means 以外の重心を用いるクラスタリング手法を検討する必要がある。

現在はすべての単語に対して一定の大きさと色で表示しているが、単語の出現頻度などによって、大きさや色を変えることで、より効率的な探索ができると考える。

進捗

平井さんは k-means という手法で実行を行っていた。その部分を k-medoids という手法で実行できるようにした。

2D グラフに対して機能を追加した。

卒論の構成を考えた。

提案手法, クラスタリング手法の違い

4/1

K-means

k-means ではクラスターの中心は「centroid」と呼ばれ、クラスター内のすべてのポイントの平均として計算される。

「centroid」は実際のデータポイントである必要はなく、仮想的な点となることが多い。

K-medoids

k-means ではクラスターの中心は「medoid」と呼ばれ、実際のデータポイントから選ばれる。

「medoid」はクラスター内の他のすべてのポイントとの平均距離が最小となるデータポイント。

提案手法, クラスタリング手法の違い

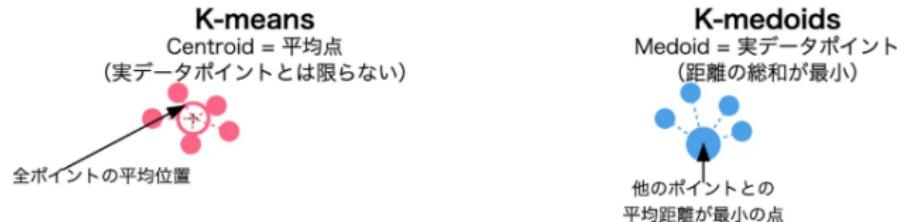
5/1

メリットとデメリット

k-means では外れ値があるとそれに引っ張られて中心がずれることがあるが、k-medoids では実際のデータ点をクラスタ中心に選ぶので外れ値に強くなる。

k-means では、計算効率がよい、k-medoids では計算コストが高く、大規模なデータでは効率が悪くなる。

K-meansとK-medoidsの中心点選択の違い



両方の方法でデータが同じクラスターに分類されていても、中心点の定義と位置が異なります

図 2: 画像 A

先行研究との比較

6/1

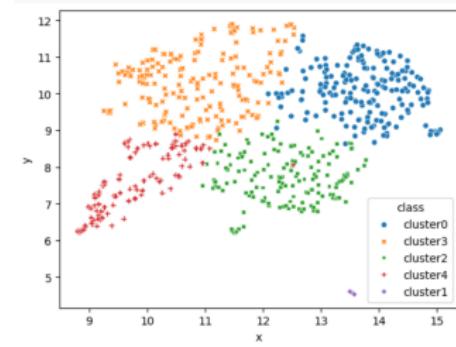


図 3: 平井さん

ChatGPTに返答させる

大本命ですね。

これに関しては

<https://note.com/negipayoc/n/n88189e590ac3>

こちらの記事を参考にしました。

記事を見てください。多分わかります。

プログラミング以外に必要なこととしてAPI keyを作成する必要があります。

[APIリファレンス](#)でサインインして、その後作ることができます。

API使用にお金がかかりますが、18ドル分以内なら無料で試せるので実質無料です。

有料になってしまっても大体日本語で1000文字で5円くらいの価格だったはずなので構手にはならないでしょう。

図 4: 今回の

追加したもの

7/1

既存の 2D グラフに追加したもの

現在のシステムでは、すべての特許が「等しい点」として扱われている。これに「重要な特許」と「そうでない特許」を区別する機能を追加。

特許の重要度を定量化し、可視化の際にノードの大きさで表現。

データ収集部に以下の指標を追加取得させる

被引用数：他の特許からどれだけ引用されているか（基本特許ほど多い）。

請求項の数：権利範囲の広さの指標。

ファミリ数：何カ国に出願しているか（コストを掛けている=重要度が高い）。

これらを重み付けして「重要度スコア」を算出。

メリット：ユーザーはマップを見た瞬間に「このクラスターの中心にある重要な特許はどれか」を一目で把握できるようになり、探索効率が向上する。

追加したもの

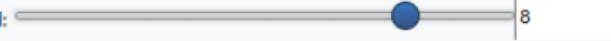
8/1

nodes

borderWidth:



borderWidthSelected:



color

border:



background:



highlight

border:



background:



hover

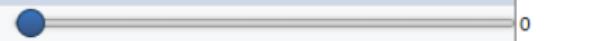
border:



background:



opacity:



fixed

x:



y:

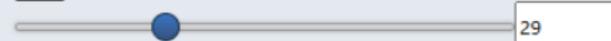


font

color:



size:



追加したもの

9/1

physics

enabled:



forceAtlas2Based

theta:



gravitationalConstant:



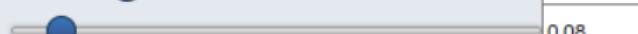
centralGravity:



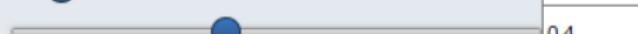
springLength:



springConstant:



damping:



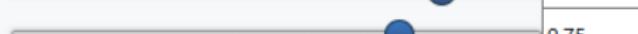
avoidOverlap:



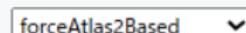
maxVelocity:



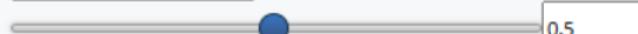
minVelocity:



solver:



timestep:

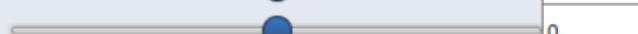


wind

x:



y:



追加したもの

既存の 2D グラフに追加したもの

Nodes (ノード / 点の設定) ここでは、特許や単語を表す「点（ノード）」の見た目を設定。

borderWidth / borderWidthSelected:

点の枠線の太さ。通常時と、クリックして選択した時の太さを指定。

color (border, background, highlight, hover):

点の色設定。通常時、選択時 (highlight)、マウスを乗せた時 (hover) の枠線と背景色を決める。

opacity: 透明度です (0~1)。

font (color, size, face, background...): 点に表示されるラベル（単語や特許番号）の文字設定。文字色、サイズ、フォント (arial など)、背景色などを変更できる。

追加したもの

追加したもの

scaling (min, max): 重要, これが今回の「特許評価スコアリング」に直結する設定.

プログラムで設定したスコア (value) に基づいて, 点がどれくらい大きくなるかの範囲を指定する.

min: スコアが低い点の最小サイズ.

max: スコアが高い点の最大サイズ.

shape: 点の形です (ellipse=楕円, circle=円, box=四角など). 現在は ellipse になっている.

size: スケーリングを使わない場合の基準サイズ.

Physics (物理演算 / 動きの設定) ここでは, ネットワーク図がどのように広がり, 配置されるかの「物理法則」を設定する.

enabled: チェックを入れると物理演算が有効になり, ノードがうねうねと動いて最適な位置を探す. 外すと動きが止まる.

solver: 計算アルゴリズムの種類. 現在は forceAtlas2Based が選択されている. これは大規模なネットワークを見やすく広げるのによく使われるモデル.

追加したもの

12/1

追加したもの

`forceAtlas2Based` の詳細設定:`gravitationalConstant` (引力定数): ここではマイナス値 (-50) になっている。これは「反発力」を意味し、ノード同士が磁石の N 極と N 極のように反発し合う。値を小さく (マイナスを大きく) すると、ノード同士がより離れようとする。

`centralGravity` (中心引力): グラフ全体が画面の中心に引っ張られる力。これが弱いとグラフが画面外へ飛んでいってしまうことがある。

`springLength` (バネの長さ): つながっているノード同士 (共起関係にある単語同士) の距離の基準。

`springConstant` (バネ定数): つながりの強さ。高いと、つながっているノード同士がギュッと近くに引き寄せられる。

`avoidOverlap`: 1 に近づけるほど、ノード同士が重ならないように調整される。文字が読みづらい場合に有効。

追加したもの

どう活用するか？

この画面でスライダーを動かすと、リアルタイムでグラフが変化する。
見やすい配置を探す: 「ごちゃごちゃしている」と思ったら,
`gravitationalConstant` をもっとマイナスにする（左へ動かす）か、
`springLength` を大きくしてみると、全体が広がる。

スコアの差を強調する: 重要な特許をもっと目立たせたい場合は、1枚目の
`scaling` の `min` を小さく、`max` を大きくすると、大小のメリハリがつく。
設定を保存する: ブラウザ上で調整して「これが見やすい！」という設定が
見つかったら、その一番下（画像には写っていませんが）に `generate options` というボタンを押すと設定コードが表示されるので、それを
Python コード (`appli4.py`) に反映させると、最初からその綺麗な形で表示
されるようになる。

ハイパーパラメータ テストデータ

14/1

特許評価・スコアリングに関するパラメータ（新規追加部分）

今回追加した「重要度スコア」を決定するパラメータ。ここを調整すると、どの特許を「大きく表示するか」の基準が変わる。

パラメータ名	設定値 (現在)	箇所 (行数 は目安)	説明	調整の指針
請求項の重み	0.7	scrap 関数内	重要度スコアにおける「請求項の数」の影響度。	権利範囲の広さをより重視したい場合は 0.8 や 0.9 に上げる。
文字数の重み	0.3	scrap 関数内	重要度スコアにおける「説明文の長さ」の影響度。	記述の詳しさを重視したい場合は数値を上げる（合計が1.0になるように調整）。
スコア倍率	180	scrap 関数内	正規化したスコア (0~1) をピクセルサイズに変換する倍率。	全体的に点を大きくしたい場合は 200 や 300 に増やす。
ベースサイズ	20	scrap 関数内	スコアが0の特許の最小サイズ。	小さすぎて見えない場合は 30 や 50 に増やす。

図 7: 画像 A

ハイパーパラメータ テストデータ

15/1

機械学習・次元圧縮モデルのパラメータ

分析の「質」そのものを決める最も重要なパラメータ群。

パラメータ名	設定値（現在）	箇所	説明	調整の指針
SBERTモデル	<code>sonoisa/sentence-bert-base-ja-mean-tokens</code>	scrap 関数冒頭	文章をベクトル化する学習済みモデル。	より特許に特化させるなら <code>tohoku-nlp/bert-base-japanese-sci</code> などに変更検討。
UMAP <code>n_neighbors</code>	5	scrap 関数内	次元圧縮時に「近くにある」と見なす点の数。	重要: 5 は局所的な構造を重視（細かい分類）。15 や 50 に増やすと大局的な構造（広い分類）が見えやすくなる。
UMAP <code>min_dist</code>	0.0 (デフォルト)	scrap 関数内	点同士の最低距離。	0.1 程度に上げると、点が密集しすぎず見やすくなる場合がある。
Metric	<code>cosine</code>	scrap 関数内	文章間の類似度を測る計算式（コサイン類似度）。	通常は <code>cosine</code> でOK。 <code>euclidean</code> に変えると距離の意味が変わる。

図 8: 画像 A

クラスタリング（分類）のパラメータ

特許をいくつのグループに分けるかを制御する。

パラメータ名	設定値（現在）	箇所	説明	調整の指針
クラス数探索範囲	2 ~ 10	<code>for i in range(2, 11):</code>	最適なクラス数を探す範囲。	特許数が多い（1000件以上）場合は、上限を15や20に広げたほうが良い分類ができる可能性がある。
乱数シード	指定なし / 42	KMeans / UMAP	結果の再現性を固定する数値。	毎回結果が変わるので防ぐには、 <code>random_state=42</code> などを明示的に指定すべき。

図 9: 画像 A

ハイパーパラメータ テストデータ

17/1

可視化・グラフ表示のパラメータ

出力される画像の見た目を制御。

パラメータ名	設定値（現在）	箇所	説明	調整の指針
プロットサイズ	(10, 8)	plt.subplots	散布図画像の大きさ（インチ）。	文字が潰れる場合は (15, 12) など大きくする。
点の透明度	0.7 (<code>alpha</code>)	sns.scatterplot	点の透け具合（0～1）。	点が重なりすぎて見にくい場合は 0.5 程度に下げる。
PyVis Solver	<code>force_atlas_2based</code>	xygraph 関数	ネットワーク図の物理演算モデル。	大規模データなら <code>barnes_hut</code> の方が軽快に動く場合がある。

図 10: 画像 A

第3章、第4章

先行研究との比較として、平井さんの結果との比較を第3章に入れた。3Dグラフで同じ単語を検索して出てきた場面を比較した。

調べたハイパーパラメータについて第4章に入れた。次にテストデータを調べて比較する必要がある。

実験

実行時間を短縮することは難しいとわかった。

openAI の導入について

先生との話の中で出てきた openAI の API の導入について。

対話型分析機能:

画面右下にチャットボックスを設置。

「クラスタ 3 とクラスタ 5 の違いは何?」と聞くと、AI が両方の特許データを比較して、「クラスタ 3 はハードウェア制御、クラスタ 5 はソフトウェア処理に関する特許です」のように答えてくれる機能。

問題点

コスト: OpenAI API は有料。大量のデータをループで処理すると課金が発生する。

しかしプログラムの性質上少しのデータで実行するのが難しいため方法を検討する必要がある。

処理時間: API 呼び出しには 1 回あたり数秒かかる。全てのクラスタを処理すると待ち時間が長くなる。

今のままでも実行時間が長いという問題があるのでさらに長くなってしまうのはどうすべきか。

先行研究との比較

20/1



図 5.3: 出力された3D グラフ

図 11: 平井さん



図 12: 今回の