Michal Najman

Topic Clustering

Jednotlivé word cloudy témat reprezentuji binárním stromem odvozeným z dendrogramu, který vznikne hierarchickým shlukováním (hierarchical clustering analysis - HCA). Jinými slovy, word cloud je binární strom témat.

K výpočtu jsou použity knihovny *numpy* a *scipy*.

Algoritmus

Vstup: N příspěvků, které jsou označena tématy. Témat je celkem K.

Výstup: Seznam témat a jejich váha; binární strom reprezentující hierarchický cluster.

1. Seznam příspěvků s detekovanými tématy je převeden do maticové reprezentace o rozměrech KxN.
2. Dále je vypočtena matice vzdáleností jednotlivých témat, resp. její vektorová neredundantní forma o délce $\binom{K}{2}$.
3. Hierarchickým shlukováním je vytvořen dendrogram ve vektorové reprezentaci.
4. Vektorová reprezentace dendrogramu je převedena na binární strom.

Výběr metody

Problematika se zjevně týká clusteringu objektů, pro něž známe matici vzdáleností (tedy clustering nad grafem, kde vzdálenosti jsou váhami hran). Byť tento problém lze řešit i jinými algoritmy, vybral jsem klasický iterativní shlukovací algoritmus, který vytvoří hierarchii clusterů slučováním množin objektů s nejmenším vzdáleností (konkrétně použita funkce *scipy.cluster.hierarchy.linkage*).

Vzdálenosti mezi tématy i a j definuji jako podíl četností (tzv. bitwise similarity), konkrétně:

$$d(i,j) = \frac{n_{01} + n_{10}}{n_{01} + n_{10} + n_{11}}$$

kde čitatel je počet příspěvků, v nichž je obsaženo jedno, nebo druhé téma (výlučně, tedy ve smyslu *xor*), a jmenovatel je počet příspěvků, v nichž je obsaženo jedno nebo druhé téma (ve smyslu *or*).

Pro výpočet vzdáleností mezi shluky existuje nespočet metod, z nichž jsem nakonec vybral dvě, které na zadaném datasetu nejlépe vybírali shluky v souladu s mou intuicí. Jedná se o metodu *ward* a metodu *average*. První jmenovaná slučuje clustery tak, aby minimalizovala empirický rozptyl, druhá metoda shlukuje podle průměrné vzdálenosti prvků ve shlucích. V konečné verzi je použita metoda *average*.

Reprezentace dat

Témata v příspěvcích: Maticová reprezentace témat v příspěvcích $A \in \{0,1\}^{K \times N}$ značí, zda k-té téma bylo detekováno v n-tém příspěvku:

$$a_{k,n} = \begin{cases} 1, & \text{téma } k \text{ je v } n\text{-tém příspěvku} \\ 0, & \text{jinak} \end{cases}$$

Sestavení této reprezentace trvá $O(NK)$. Paměťová náročnost je $\Theta(NK)$, jelikož současná verze matici ukládá explicitně.

Vzdálenosti témat: Vektor vzdáleností má délku $\binom{K}{2}$ a jeho sestavení je z hlediska časové složitosti nejnáročnější, konkrétně $O(NK^2)$. Lze tedy vidět, že škálování algoritmu bude nejvíce citlivé na celkový

počet témat. Bohužel, chceme-li požít metodu, která vyžaduje k výpočtu vzdálenost mezi tématy, tento výpočet bude „úzkým hrdlem”.

Strom témat: Co se týče dedrogramu, zvolená shlukovací metoda jej generuje do reprezentujícího vektoru v čase $O(K^2)$. Nicméně tato reprezentace je neintuitivní, a proto jsem se rozhodl hierarchickou povahu shluků převést na binární strom, jehož listy jsou témata a strukturu určuje vztah témat v rámci hierarchického clusteru. Binární strom je reprezentován jako spojový binární strom (tedy každý uzel si drží reference na levého a pravého potomka). Sestavení stromu trvá $O(K \log K)$.

Formát výstupu

Výstupem algoritmu je, jak již bylo zmíněno, binárním strom a váhy jednotlivých slov (pro výpočet velikosti fontu v rámci word cloudu). Výstup má následující JSON formát (kompletní vzorový vstupu a výstupu jsou ukázány v příloze na konci dokumentu):

```
{
      "words": [
            {
                  "name": "t1",
                  "weight:": 0.6,
                  "word_id": 0
            },
            …
      ],
      "hierarchical_cluster": *Node*
}
```

Objekt **Node** je definován následovně, pokud se jedná o list:

```
{
      "leaf": true,
      "word_id": 0
}
```

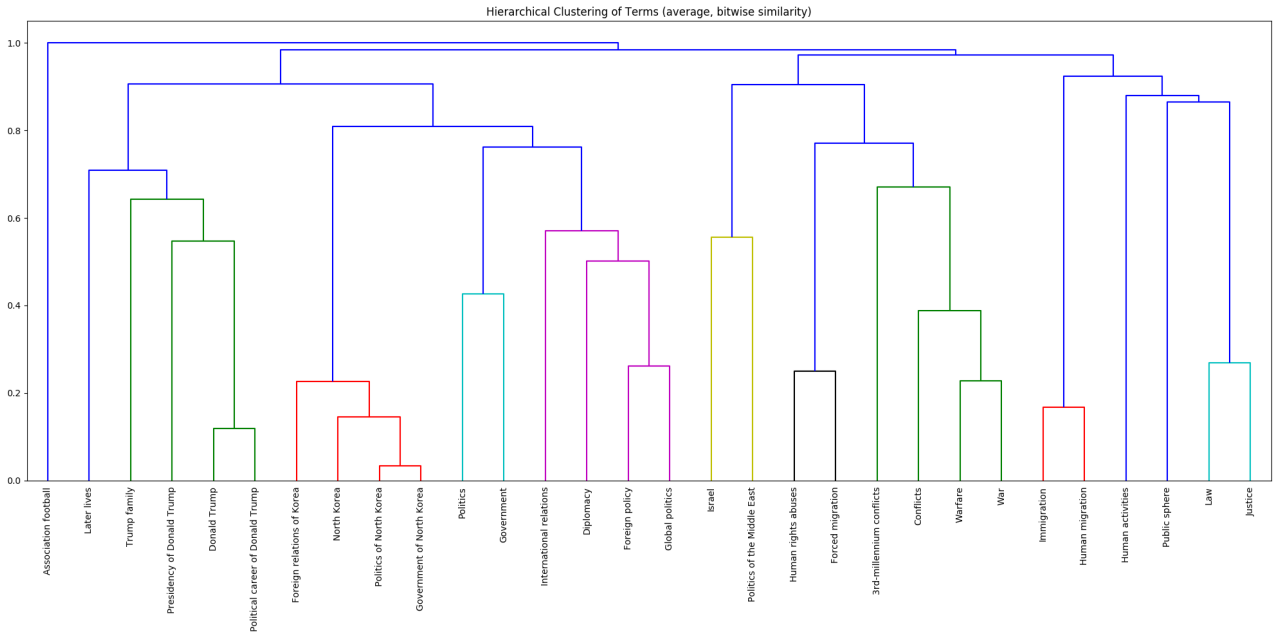
A pokud se o list nejedná, je **Node** definován takto:

```
{
    "leaf": false,
    "left_child": *Node*,
    "right_child": *Node*,
    "cluster_size": 2,
    "children_distance": 0.4
}
```

Při sestavování word cloudu dle zadání se na začátku zobrazí všechna slova v jednom cloudu – což odpovídá kořenu stromu. Při kliknutí na word cloud se tento rozpadne na dvě množiny slov, odpovídající slovům v listech levého a pravého podstromu.

Co se týče velikosti fontu slov, lze k výpočtu použít váhu *weight*, která odpovídá relativní frekvenci tématu v příspěvcích. Pro vzdálenost jednotlivých word cloudů lze použít hodnotu *children_distance* značící vzdálenost mezi podstromy daného uzlu.

Je nutno podotknout, že formát výstupu jsem vybral s ohledem jak na strojovou, tak lidskou čitelnost. Nicméně je zřejmé, že pro konkrétní algoritmus ke tvorbě word cloudu se mohou hodit jiné reprezentace dat. Kód je psán tak, aby nebylo obtížné tuto reprezentaci změnit (konkrétně metoda *ClusterEncoder.to_dict(cluster)*).



Diskuze vybraného řešení

Řešení je postaveno na numpy.ndarray maticích a shlukovacích funkcích v knihovně scipy (*scipy.cluster.hierarchy.linkage)*. Záměrně jsem nechtěl používat jiné méně optimalizované knihovny (např. sci-kit learn, která ale také nabízí velmi mocné nástroje pro práci s clustery).

Ačkoliv lze „kvalitu" clusterů vyhodnocovat strojově (např. entropií apod.), tomuto postupu jsem se vyhnul a dendrogram posuzoval *zkušeným okem odborníka*. Na obrázku je ukázán dendrogram hierarchického shluku pro zadaná vstupní data. Dle mé intuice jsou témata poměrně zdárně sestavena do korektního vztahu a jejich vzdálenosti odpovídají. Barvy značí význačné clustery témat, čehož by se ve vizualizaci dalo jistě také využít.

Vzdálenosti témat se počítají bitwise podobností (převzato z Social Big Data Mining od Hiroshi Ishikawi) a vzdálenost shluků je počítána metodou *average*, tedy průměrenou vzdáleností témat.

Ověření správnosti

Zřejmě, k plnému ověření správnosti kódu je zapotřebí napsat unit testy a sadu korektních i nekorektních příkladů vstupních dat. Nicméně, kvůli deadlinu jsem se rozhodnul správnost kódu ověřit jen namátkou v Jupyter notebooku. Prototyp řešení jsem psal v notebooku *notebooks/prototype.ipynb* a řešení jsem postupně testoval v notebooku *notebooks/example.ipynb*. Pro daný účel je tento postup dostatečný, v případě deploye do produkce by ale unit testy minimálně na nekorektní vstup potřeba určitě byly.

Škálovatelnost

Výpočet vzdáleností témat je nejnáročnější operací a bohužel ji řádově (myšleno asymptoticky) nelze zrychlit. Nicméně triky pro částečné zrychlení by se pro velká K daly vymyslet. Například počítat vzdálenosti jen mezi tématy, která se spolu objevila v alespoň jednom z příspěvků. Nicméně lze předpokládat, že seznam témat nebude příliš dlouhý a jeho distribuce bude mít tzv. long-tail, tedy spousta témat bude zastoupena jen řídce. Pro zmenšení K lze tato témata z výpočtu vyřadit, jelikož ve vizualizaci nebudou hrát roli.

Výzvou pro škálování je počet příspěvků, tedy číslo N. Vybraný postup je však asymptoticky lineární v N, tedy nejmenší možný, vezmeme-li v potaz povahu vstupních dat.

Na čase a paměti by se dalo ušetřit při sestavování maticové reprezentaci příspěvků a následné počítání vzdáleností témat. Zde se dějí tyto procedury: seznam příspěvků iteruji dvakrát, následně explicitně ukládám matici tvaru $K \times N$ (jedná se o *numpy.ndarray* nad proměnnými typu *boolean*) a poté vytvořím vektor vzdáleností témat v čase $O(NK^2)$.

Tento postup jsem vybral, jelikož je pro danou škálu vstupních dat programátorsky nejjednodušší. Pro velká N a K by bylo vhodnější počítat vzdálenosti přímo ze vstupního formátu a nepřevádět příspěvky na matici booleanů – témat bude vždy výrazně méně než příspěvků, a tedy by se po přepsání výpočtu výrazně zmenšila paměťová složitost z $\Theta(NK)$ na $\Theta(K^2)$.

„Úzké hrdlo" leží také v samotném clustering algoritmu, jehož implementace běží v čase $O(K^2)$. Bohužel, zde již není prostor ke zlepšování – implementace algoritmu v knihovně scipy pro vektor vzdáleností je poměrně „vychytaná". Samozřejmě, alternativou by mohly být jiné metody než klasický clustering. Nicméně se obávám, že rychleji než $O(K^2)$ žádný rozumný algoritmus stavící strom závislostí nepoběží. Pokud by data byla natolik veliká, že je $O(K^2)$ vysoké, lze problém relaxovat. Například: stavět strom jen do určitě hloubky; nebo použít Monte-Carlo metodu, která by strom stavěla aproximativně apod.

V neposlední řadě lze mírného zrychlení dosáhnout vybráním jiné reprezentace hierarchických shluků.

Spuštění programu

Pro spuštění programu pro zadaná vstupní data stačí spustit v terminálu (working directory musí být kořen adresáře projektu):

```
scripts/example.sh
```

A pro jakákoliv vstupní data použijte následující příkaz a nahraďte vstupní a výstupní soubor:

```
python3 source/main.py *input.json* *output.json*
```

Zabalit projekt do docker contraineru a spustit v cloudu jako službu je otázka několika minut.

Zdrojové kódy projektu jsou na mém gitu: bit.ly/word-cloud-clustering.

Příloha - vzorový vstup a výstup

Pro vzorový vstup:

```
[
    {
        "postId": "1",
        "topics": [
            "Apple"
        ]
    },
    {
        "postId": "2",
        "topics": [
            "Apple",
            "Google"
        ]
    }
]
```

Má výstup následující formát:

```
{
      "words": [
            {
                  "name": "Apple",
                  "weight:": 1.0,
                  „word_id": 0
            },
            {
                  "name": "Google",
                  "weight:": 0.5,
                  "word_id": 1
            }
      ],
      "hierarchical_cluster": {
            "leaf": false,
            "left_child": {
                  "word_id": 0,
                  "leaf": true
            },
            "right_child": {
                  "word_id": 1,
                  "leaf": true
            },
            "cluster_size": 2,
            "children_distance": 0.5
      }
}
```