دانشگاه صنعتي اميركبير

(پلی‌تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

پروژه مباحثی در ریاضیات و کاربرد ها:

خوشه بندی با استفاده از تجزیه نامنفی ماتریس

نگارش

آریان فتحی

ارسلان تازیکه

گروه ۱۰

استاد درس

دکتر فاطمه شاکری

فروردین ۱۴۰۴

چکیده

در این پروژه قصد داریم الگوریتم های خوشه بندی KMeans و NMF را روی دیتاست 20 newsgroups اجرا کنیم. همچنین الگوریتم KMeans را خودمان به صورت دستی پیاده سازی کنیم و نتایج آن ها را با هم مقایسه کنیم. در ابتدا تئوری هر یک از این روش ها را با هم مرور می کنیم. سپس وارد فاز پیاده سازی می شویم و در انتها این الگوریتم های خوشه بندی از نظر معیار های خوشه بندی AMI، ARI و Silhouette با هم مقایسه می کنیم.

فهرست مطالب

صفحه

چکیده.....أ

فصل اول: تئوری تجزیه نامنفی ماتریس و خوشه بندی.....3

فصل دوم: پیاده سازی.....6

2-1- معیار آماری TF-IDF.....8

2-2- معیار های AMI و ARI.....16

2-3-1- معیار AMI.....16

2-3-2- معیار ARI.....18

نتایج الگوریتم های خوشه بندی و تحلیل آنها.....19

لینک پروژه.....21

فصل اول: تئوری تجزیه نامنفی ماتریس [1]و خوشه بندی

در این قســمت تنها به ارتباط این تجزیه و بحث خوشــه بندی می پردازیم چرا که ســایر مســائل همچون الگوریتم های محاسـبه تجزیه نامنفی در جزوه درس دکتر شـاکری به طور کامل بیان شـده است.

خوشه بندی یکی از تسک های معروف یادگیری ماشین است که جزو دسته الگوریتم های یادگیری بدون ناظر[2] قرار می گیرد.

در این پروژه ما این تســک را توســط الگوریتم های K-Means و تجزیه نامنفی ماتریس انجام می دهیم و عملکرد و نتایج آنها را خواهیم دید.

الگوریتم خوشــه بندی K-Means بدین صــورت اســت که یک افراز اولیه برای داده ها درنظر می گیرد. سپس میانگین خوشه ها که توسط فرمول زیر محاسبه می شود را برای تمامی خوشه بدست می آورد:

میانگین خوشه j-ام: $$m_j = \frac{1}{n_j}\sum_{v\in\pi_j} X_v$$

که n_j تعداد اعضای π_j است.

---

[1] Non-Negative Matrix Factorization

[2] Unsupervised Learning

سپس معیار کیفیت خوشه بندی را برای این خوشه بندی حساب می کند. دقت کنیم این معیار بر خلاف اسم ظاهری آن، در واقع تابع زیان[3] خوشه بندی درنظر گرفته می شود و هدف الگوریتم K-Means پیدا کردن افرازی از داده ها است که این تابع را کمینه کند. این تابع به صورت زیر تعریف می شود:

کیفیت خوشه‌بندی Π:

$$Q(\Pi) = \sum_{j=1}^{k} q_j = \sum_{j=1}^{k} \sum_{v \in \pi_j} \|X_v - m_j\|_2^2$$

که هر یک از q_j ها، وابستگی خوشه j-ام نامیده می شوند که به صورت زیر تعریف می شوند:

وابستگی خوشه j-ام: $$q_j = \sum_{v \in \pi_j} \|X_v - m_j\|_2^2$$

هر قدر بردارهای خوشه j-ام به میانگین m_j نزدیک‌تر باشد، q_j کوچک‌تر است.

به عبارتی با توجه به تعریف تابع کیفیت خوشه بندی بر اساس این مفهوم و سعی ما در کمینه کردن این تابع در الگوریتم K-Means ، به دنبال خوشه بندی هستیم که تا جای ممکن داده های در هر خوشه به میانگین نزدیک باشند و پراکندگی یا به اصطلاح آماری واریانس داده ها در هر خوشه کم باشد(استحکام خوشه ها بیش تر می شود). در چنین حالتی خوشه های ما dense تر

[3] Loss Function

خواهند بود و خوشه بندی بهتری روی داده ها انجام خواهد شد.(با توجه به درنظر گرفتن این تابع زیان)

در الگوریتم K-Means، بعد از خوشه بندی اولیه برای هر داده نزدیک ترین میانگین خوشه را پیدا می کنیم و آن داده را به خوشه متناظر با آن منتقل می کنیم. یعنی در این مرحله ما افراز جدیدی از داده ها به خوشه ها خواهیم داشت. حال میانگین خوشه های جدید و همچنین کیفیت این خوشه بندی را بدست می آوریم. اگر اختلاف کیفیت دو خوشه بندی متوالی از مقدار تعیین شده[4] ما کمتر شد، می توانیم الگوریتم خوشه بندی را متوقف کنیم و آخرین خوشه بندی را به عنوان خوشه بندی مطلوب در نظر بگیریم. در غیر این صورت به مرحله آپدیت از الگوریتم برمی گردیم و سایر مراحل الگوریتم را ادامه می دهیم.

حال به بیان کاربرد تجزیه نامنفی ماتریس در خوشه بندی می پردازیم.

به طور خلاصه اگر بخواهیم تعدادی داده را به تعدادی خوشه افراز کنیم، آن داده ها را به عنوان ستون های یک ماتریس درنظر می گیریم و تجزیه نامنفی را روی آن ماتریس انجام می دهیم(دقت کنیم تعداد خوشه به عنوان تعداد ستون های ماتریسW و تعداد سطر های ماتریس H در تجزیه نامنفی درنظر گرفته می شوند)

$$\boldsymbol{X \simeq WH,\ \ k \leq min\{m,n\}}$$

سپس هر یک از ستون های ماتریس X را توسط ترکیب خطی از ستون های W میسازیم که ضرایب همان سطر های ستون متناظر با داده از ماتریس H هستند. خوشه مربوط به داده را متناظر با شماره سطر مربوط به بزرگترین ضریب این ترکیب خطی در نظر می گیریم. یا به عبارتی بزرگترین درایه در ستون مربوطه از ماتریس H.

---

[4] Threshold

فصل دوم: پیاده سازی

در این بخش قسمت های مهم و خواسته شده از پیاده سازی را شرح خواهیم داد.

در ابتدا داده را به همان شـکلی که به صـورت راهنما گفته شـده بود می خوانیم. در واقع داده های مربوط به دسته بندی [5]هایی که مدنظر بود را از قسمت آموزشی دیتاست می خوانیم:

```
from sklearn.datasets import fetch_20newsgroups

def load_newsgroup_data(categories=None, subset='train'):
    """
    Loads the 20 Newsgroups dataset for the specified categories and subset.

    Parameters:
        categories (list): List of category names to load. If None, loads all categories.
        subset (str): Subset of data to load. Options: 'train', 'test', or 'all'.

    Returns:
        sklearn.utils.Bunch: The loaded dataset, with attributes like data and target.
    """
    newsgroups = fetch_20newsgroups(categories=categories, subset=subset)
    return newsgroups

# Example usage
categories = ['comp.graphics', 'talk.politics.guns', 'alt.atheism', 'sci.med', 'sci.space']
data = load_newsgroup_data(categories)
print(len(data.data))
print(len(data.target)) # Ground truth labels 0: comp.graphics, 1: talk.politics.guns, 2: alt.atheism, 3: sci.med, 4: sci.space
```

```
2797
2797
```

این تابع داده را به شـــکل یک شـــئ از کلاس Sklearn.utils.bunch بر می گرداند که شـــبیه به دیکشـنری اسـت و آرایه از داکیومنت ها و آرایه های از لیبل آنها را به شـکل عدد صـحیح در خود دارد. اگر attribute دیتا از آن را درنظر بگیریم، داکیومنت ها را خواهیم داشـــت که همان طور که میبینم تعداد 2797 داکیومنت در این دیتاسـت وجود دارد. همچنین به همین تعداد هم لیبل وجود دارد(سایز آرایه مربوط به لیبل ها) که منطقی است.

[5] Categori

در قســـمت پیش پردازش داده های متنی، با اســـتفاده از کتابخانه NLTK، علائم نگارشی، اعداد و stopword ها از داکیومنت ها حذف کردیم. دقت کنیم ابتدا داکیومنت ها را از لهجه های مختلف لاتین به انگلیسی برگرداندیم توسط تابع ()remove-accents.

```
def remove_accents(text):
    return unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8')

# Example usage
remove_accents("café")
```

```
'cafe'
```

```
[ ] lemmatizer = WordNetLemmatizer()
    STOP_WORDS = set(stopwords.words('english'))

    preprocessed_data = []
    for doc in tqdm.tqdm(data.data, desc="Processing data"):
        doc = remove_accents(doc)
        doc_tokens = nltk.word_tokenize(doc)
        tokens = [
            lemmatizer.lemmatize(word.lower().strip())
            for word in doc_tokens
            if (
                word.isalpha() and
                word.lower() not in STOP_WORDS and
                word not in string.punctuation
            )
        ]

        preprocessed_data.append(" ".join(tokens))
```

در تکه کد بالا، ابتدا داکیومنت ها را به توکن ها تقســیم می کنیم و ســپس هر یک از توکن ها را به ریشــه اصــلی کلمه برمی گردانیم. توکن هایی که عدد نیســتند، stopword هســتند و علائم نگارشی هســتند را حذف می کنیم. خروجی لیســتی از داکیومنت ها اســت که هر یک لیســتی از توکن ها هستند.

مثال:

```
[ ] print("Example of a preprocessed doc(index 0):", preprocessed_data[0])
    print("Example of a preprocessed doc(index 100):", preprocessed_data[100])

Example of a preprocessed doc(index 0): mhamilto lawnmowerman subject atf burn dividian ranch survivor keywords nata thing matthew hamilton organizat
Example of a preprocessed doc(index 100): dprjdg john grasham subject give billion first organization arco oil gas company line keithley craig keithl
```

2-1- معیار آماری TF-IDF

این معیار مخفف Term Frequency-inverse Document Frequency است. یک معیار آماری است برای اینکه اهمیت یک کلمه در یک داکیومنت را نسبت به مجموعه کلی داکیومنت ها ارزیابی کنیم. این معیار از دو مفهوم TF و IDF تشکیل شده است که هریک به شکل های زیر تعریف می شوند:

$$\mathrm{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

معیار TF در واقع میزان اهمیت یک کلمه یا ترم را در یک داکیومنت نشان می دهد.

$$\mathrm{IDF}(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing term } t}\right)$$

ترم ها یا کلمه هایی که در تعداد زیادی از داکیومنت ها ظاهر می شوند، مقدار IDF برای آنها کوچک تر است. به این معنا که انگار ارزش اطلاعاتی زیادی ندارند.

در نهایت معیار TF-IDF از حاصل ضرب این دو مفهوم به دست می آید:

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

در واقع این معیار کمک می کند که کلماتی که در یک داکیومنت مهم هستند و از طرف دیگر در بین داکیومنت ها مشترک نیستند و به نوعی می توان آنها را جزو مشخصه های یک داکیومنت درنظر گرفت را پیدا کنیم. از طرف دیگر خیلی از کلماتی که در بین تعداد زیادی از داکیومنت ها مشترک

هستند، ممکن است داده های نویزی باشند و به وسیله این معیار، می توانیم امتیاز و وزن کم تری برای آنها قائل شویم. در تسک ما که خوشه بندی است و دوست داریم داده هایی که در یک خوشه قرار می گیرند، ویژگی های مشترک زیادی داشته باشند و به هم شبیه باشند، استفاده از این معیار بسیار کمک کننده است. داکیومنت هایی را که در کلماتی مشترک هستند و از طرفی آن کلمات در داکیومنت های دیگر(خوشه های دیگر) وجود ندارد می توان کاندید تشکیل یک خوشه در نظر گرفت.

در تکه کد زیر با استفاده از TfidfVectorizer آرایه های از داکیومنت ها را به ماتریس ترم-داکیومنت تبدیل می کنیم که هر یک از درایه های این ماتریس امتیاز های tf-idf مربوط به ترم ها در داکیومنت ها هستند.(دقت کنیم در این ماتریس سطر ها داکیومنت ها هستند و ستون ها مربوط به کلمات هستند):

Vectorizing with Tf-Idf

```
tfidf_vectorizer = TfidfVectorizer(
    analyzer='word',
    stop_words='english',
    token_pattern=r'\b[a-zA-Z]{3,}\b',  # Minimum 3 letters
    lowercase=True,                     # Convert everything to lowercase
    max_df=0.95,                        # Ignore very frequent terms
    min_df=5                            # Ignore very rare terms
)
dtm_tfidf = tfidf_vectorizer.fit_transform(preprocessed_data)
dtm_tfidf.shape
```

```
(2797, 7509)
```

همان طور که میبینیم constructor این شئ tf-idf به شکل بالا صدا زده ایم. دو پارامتر max-df و min-df نشان می دهند که دنبال کلماتی هستیم که در کمتر از ۹۵ درصد داکیومنت ها و بیش تر از ۵ درصد داکیومنت ها ظاهر شده باشند. این ست کردن اینگونه پارامتر ها باعث می شود که کلماتی که خیلی مشترک هستند بین داکیومنت ها و کلماتی که خیلی کم تکرار شده اند و ممکن است واقعا بار اطلاعاتی زیادی نداشته باشند را به عنوان ستون های ماتریس نهایی درنظر نخواهیم گرفت. همین طور آنهایی که کم تر از ۳ حرف انگلیسی- را شامل می شوند. تمامی کلمات به lower

تبدیل شـده اند. به عبارتی Democracy و democracy دوسـتون متمایز از ماتریس را در نخواهند گرفت و یک ستون برای democracy درنظر گرفته می شود.

ازآنجایی که ماتریس tf-idf حاصل ممکن است دارای درایه های صفر زیادی باشد(امتیاز های صفر برای کلمات)، در نتیجه با تکه کد زیر، این ماتریس ترم-داکیومنت را dense خواهیم کرد:

```
[14] dtm_dense = dtm_tfidf.toarray()
```

در تکه کد زیر با استفاده از تجزیه نامنفی ماتریس، ماتریس های W و H حاصله از ماتریس tf-id را به دســـت آورده ایم که ماتریس W یک ماتریس document-topic اســـت که هر درایه (i,j) آن نشـان دهنده قدرت و امتیاز topic j-ام در داکیومنت i-ام اسـت. همین طور ماتریس H هم نشـان دهنده ماتریس topic-terms اســـت که هر ســـطر آن به یک موضـــوع و هرســـتون آن به یک ترم مربوط است و اهمیت هر یک از ترم ها را در موضـوعات مختلف(خوشـه های مختلف) نشـان می دهد. حال در ماتریس W اندیس مربوط به بزرگترین درایه ی هر ســـطر(داکیومنت) ماتریس را درنظر می گیریم. یعنی آن داکیومنت در خوشـــه(مربوط به موضـــوع) متناظر با آن اندیس قرار می گیرد:

```
# number of clusters(topics)
n_topics = 5

nmf_model = NMF(
    n_components=n_topics,
    init='nndsvda',       # often speeds convergence, because of filling zero values with mean to achive more dense matrix
    random_state=42,
    max_iter=200
)

t0 = time.perf_counter()
W = nmf_model.fit_transform(dtm_tfidf)
H = nmf_model.components_
t_nmf = time.perf_counter() - t0

# For each document i, find which topic has the highest W[i, :] as score
doc_topic_labels_nmf = W.argmax(axis=1)

mem_nmf = memory_bytes(W) + memory_bytes(H)

scores_nmf = evaluate(doc_topic_labels_nmf, W, ground_truth_labels=data.target)
```

در ادامه هم سعی کردیم از tsne برای کاهش بعد ماتریس W استفاده کنیم تا داده(داکیومنت) ها در فضای دوبعدی نمایش دهیم و درک بصری بهتری از خوشه ها بدست آوریم.

```
[ ] tsne = TSNE(n_components=2, random_state=42)
    tsne_coords_reduced = tsne.fit_transform(W)
```

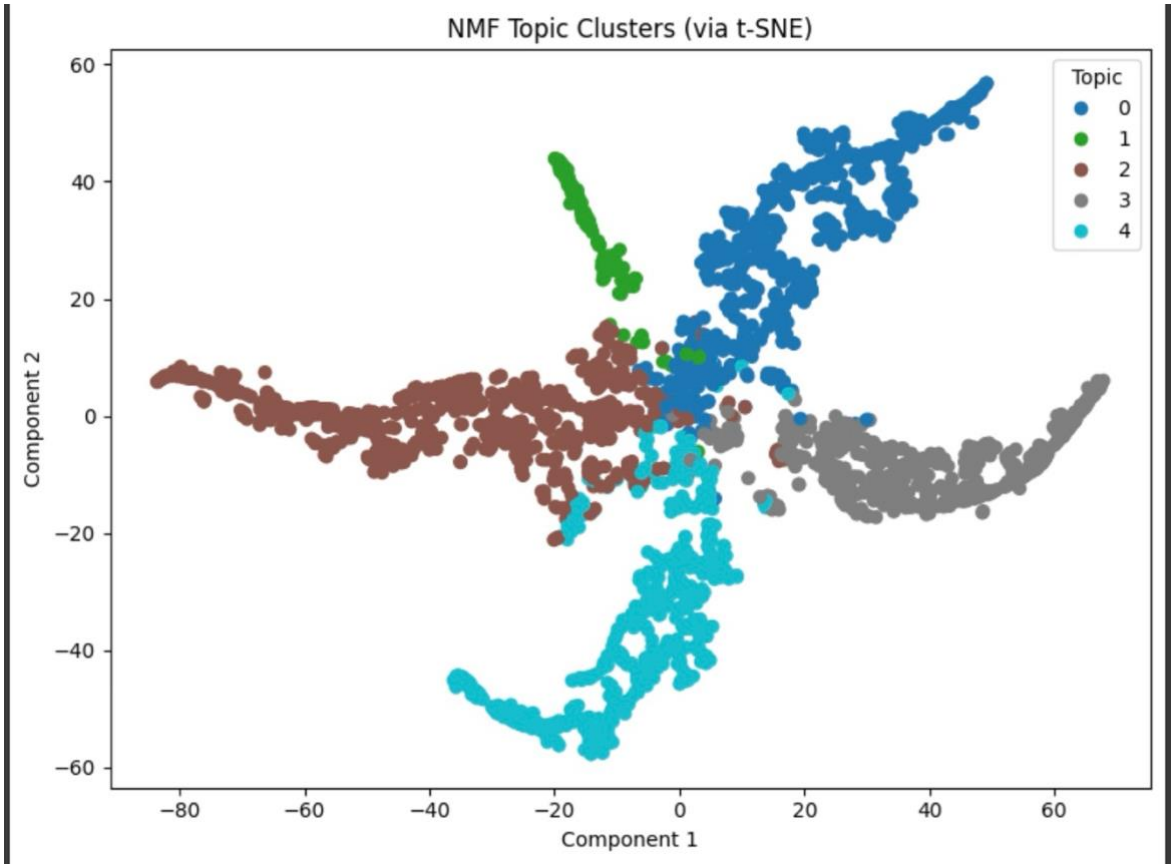
همان طور که گفتیم، ماتریس H شامل موضوعات به عنوان سطر ها و ترم ها به عنوان ستون ها است و درایه ها نیز نشان دهنده اهمیت هریک از ترم ها در هریک از موضوعات هستند. حال با استفاده از این ماتریس در کد زیر، ۱۰ ترم مهم از هریک از موضوعات را نشان می دهیم:

```
# Inspecting the top words in each topic
feature_names = tfidf_vectorizer.get_feature_names_out()
n_top_words = 10

for topic_idx, topic in enumerate(H):
    top_indices = topic.argsort()[-n_top_words:][::-1]
    top_terms = feature_names[top_indices]
    top_scores = topic[top_indices]
    print(f"Topic #{topic_idx}:")
    print("  ", ", ".join(top_terms))
```

```
Topic #0:
   gun, people, weapon, right, firearm, law, like, msg, think, crime
Topic #1:
   geb, gordon, bank, shameful, chastity, skepticism, intellect, pittsburgh, surrender, soon
Topic #2:
   file, graphic, image, thanks, program, format, university, help, color, package
Topic #3:
   god, keith, atheist, livesey, morality, moral, writes, jon, say, religion
Topic #4:
   space, henry, nasa, moon, launch, orbit, shuttle, lunar, cost, station
```

در ادامه ۲۰۰ کلمه از ۵ داکیومنت اول خوشه ی با اندیس ۰ را نشان داده ایم و می توانیم خوشه بندی انجام شده را به صورت بصری ببینیم:



شکل 1 خوشه بندی با NMF

در ادامه خوشه بندی با الگوریتم k-means که توسط خودمان پیاده سازی شده است را بررسی می کنیم. این الگوریتم را در فصـــل اول توضـــیح دادیم و در اینجا آن را با قســـمت های مختلف کد متناظر می کنیم:

```
# 1. Choosing k random docs as centroids
init_idxs = np.random.choice(n_docs, size=k, replace=False)
centroids = X[init_idxs]
```

در این تکه کـد، در واقع یـک افراز اولیـه از داده هـا(داکیومنـت هـا) درنظر می گیریم. اینجـا k تـا داکیومنت را به عنوان مراکز خوشه به طور تصادفی انتخاب می کنیم.

```python
# 1. Choosing k random docs as centroids
init_idxs = np.random.choice(n_docs, size=k, replace=False)
centroids = X[init_idxs]

for iteration in tqdm.tqdm(range(max_iters), desc="Converging"):
    # 2. Assign each point to the nearest centroid
    #    compute squared Euclidean distances
    #    shape of dists: (n_docs, k)

    dists = np.linalg.norm(X[:, None, :] - centroids[None, :, :], axis=2)

    labels = np.argmin(dists, axis=1)  # for each sample, index of closest centroid

    # 3. Update centroids as the mean of assigned points
    new_centroids = np.zeros_like(centroids)
    for cluster in range(k):
        members = X[labels == cluster]
        if len(members) > 0:
            new_centroids[cluster] = members.mean(axis=0)
        else:
            # re-initialize empty cluster to a random point
            new_centroids[cluster] = X[np.random.choice(n_docs)]

    # 4. Check for convergence
    centroid_shift = np.linalg.norm(new_centroids - centroids)
    if centroid_shift < tol:
        print()
        print(f"Converged in {iteration} iterations.")
        break

    centroids = new_centroids

return labels, centroids
```

در این تکه کد، فاصله اقلیدسی هر دیتاپوینت را از هر مرکز خوشه به دست می آوریم. سپس آن داده را به خوشه منتقل می کنیم.(لیست labels در کد بالا). در قسمت بعدی مراکز خوشه را آپدیت می کنیم و اگر خوشه ای خالی ماند، یک دیتا را رندوم در آن قرار می دهیم. حال یک خوشه بندی جدید داریم. شرط همگرایی را چک می کنیم که در صورت همگرا شدن، دیگر آپدیتی انجام ندهیم. و در انتها هم لیست مربوط به خوشه ی هر یک از داده ها و لیست شامل مراکز خوشه ها را بر می گردانیم.

در ادامه یکبار این الگوریتم را روی ماتریس W که ابعاد کمتری نسبت به ماتریس اصلی tf-idf دارد انجام می دهیم و یکبار هم روی ماتریس اصلی tf-idf.

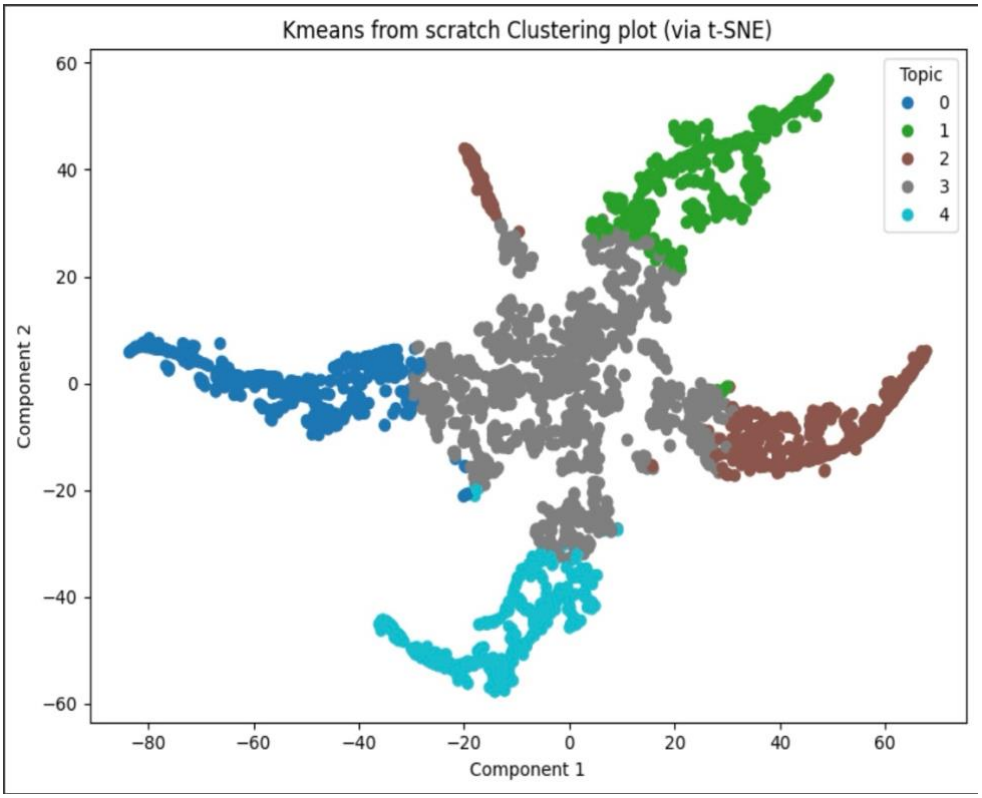
در زیر می توانید عملکرد این دو الگوریتم را مشاهده کنید:

```
Running on Reduced DTM
Converging:  15%|▌         | 15/100 [00:00<00:00, 892.14it/s]
Converged in 15 iterations.


Running on Non-Reduced DTM
Converging:  12%|▌         | 12/100 [00:11<01:25,  1.03it/s]
Converged in 12 iterations.
```

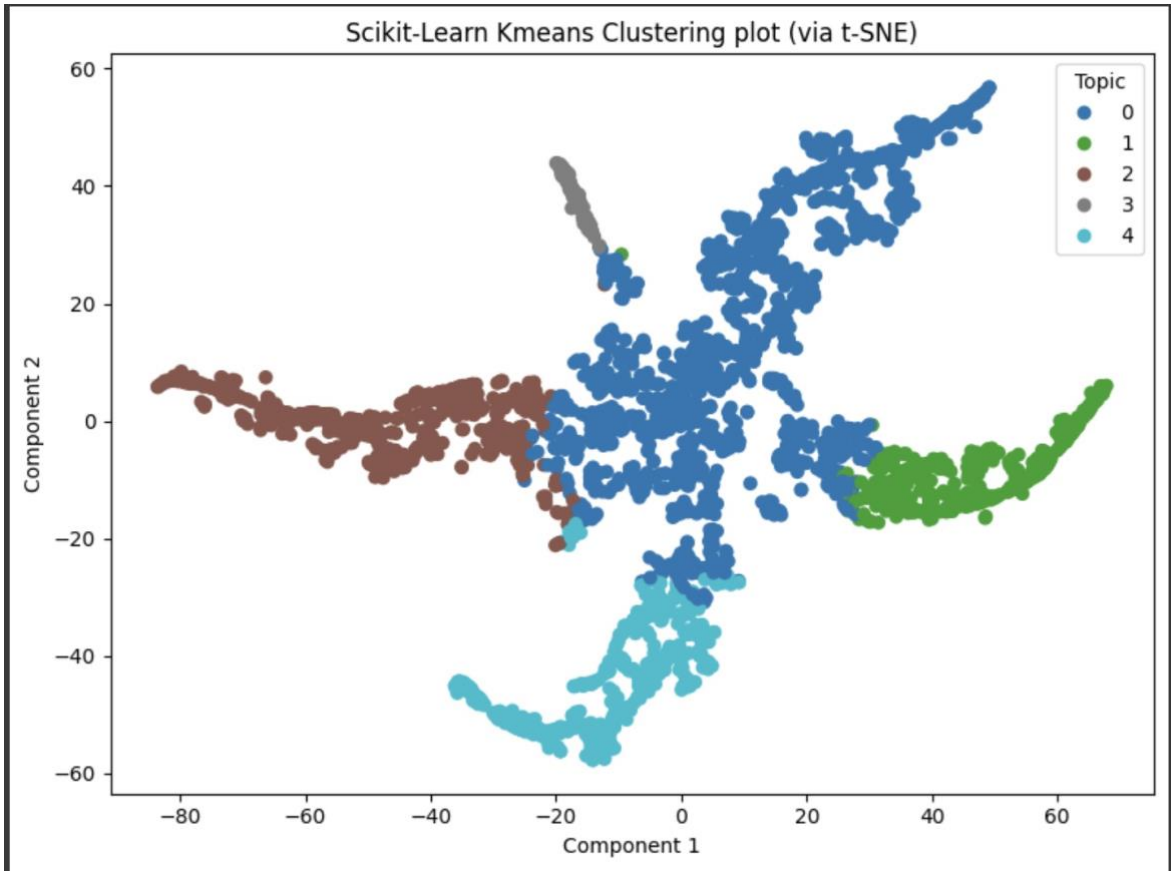
همان طور که مشاهده می کنیم، روی ماتریس non-reduced همگرایی سریع تر رخ داده است. هم چنین در هر دو حالت، با تعداد کم تر از ۲۰ تکرار الگوریتم همگرا شده است. شاید همگرایی سریع تر به این دلیل رخ داده باشد که در ابعاد بالاتر، داده ها از هم جدا تر هستند و در ابعاد پایین بحث crowding problem را داریم و همین جدا بودن بیش تر داده ها از هم، باعث می شود الگوریتم خوشه بندی k-means نیازی به تکرار های بیش تر برای رسیدن به آستانه مورد نظر نداشته باشد. البته این همیشه به این معنا نیست که هر چه ابعاد را بالا ببریم، ساختارداده ها و فاصله شان از هم بهتر حفظ می شود و ابعاد بالا همیشه خوب است ولی در این مورد می توان به این موضوع اشاره کرد.

در زیر هم می توانید خروجی خوشه بندی با الگوریتم K-Means from scratch را ببینیم:



شکل 2 خوشه بندی با الگوریتم KMeans from scratch

در ادامه هم می توانیم نتیجه حاصل از خوشه بندی با الگوریتم K-Means موجود در کتابخانه Scikit-Learn را مشاهده کنیم. دقت کنیم این الگوریتم نیز هم بر روی ماتریس W حاصل از NMF اعمال شده و هم بر روی ماتریس اصلی tf-idf:



شکل 3 KMeansبا استفاده از SKlearn

2-2- معیار های AMI و ARI

2-3-1- معیار AMI

Adjusted Mutual Information معیاری است که برای ارزیابی کیفیت خوشه‌بندی با مقایسه شباهت بین برچسب‌های واقعی یک مجموعه داده و برچسب‌های تولید شده توسط الگوریتم

خوشه‌بندی استفاده می‌شود. این معیار، نمره اطلاعات متقابل [6]را برای محاسبه گروه‌بندی‌های تصادفی تنظیم می‌کند و اندازه‌گیری دقیق‌تری از توافق ارائه می‌دهد.

اطلاعات متقابل(MI)

اطلاعات متقابل مقدار اطلاعاتی را که از یک متغیر تصادفی از طریق متغیر دیگر بدست می‌آید را نشان می دهد. در زمینه خوشه‌بندی، این معیار اندازه‌گیری می‌کند که چقدر دانستن خوشه‌بندی (برچسب‌های پیش‌بینی شده) درباره برچسب‌های واقعی به ما اطلاعات می‌دهد.

فرمول آن به شکل زیر است:

$$MI = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{n_{ij}}{n} \log \left(\frac{n \cdot n_{ij}}{n_i \cdot n_j} \right)$$

که در آن :

n_i مجموع شمارش داده ها در خوشه i از برچسب‌های واقعی است.

n_j مجموع شمارش داده ها در خوشه j از برچسب‌های پیش‌بینی‌شده است.

n تعداد کل داده ها

n_{ij} شمارش داده های در خوشه i از برچسب‌های واقعی و خوشه j از برچسب‌های پیش‌بینی‌شده است.

در واقع این معیار مقدار اطلاعاتی که دو خوشه بندی با هم به اشتراک می گذارند را اندازه می گیرد.

[6] Mutual Information

$$AMI = \frac{MI - E[MI]}{\max(H(U), H(V)) - E[MI]}$$

در معیار AMI ما اثر شباهت تصادفی دو خوشه بندی را حذف میکنیم. در واقع با کم کردن مقدار موردانتظار معیار اطلاعات متقابل از MI این کار را انجام می دهیم. حال هر چه مقدار بیش تری داشته باشد، خوشه بندی حاصل از الگوریتم خوشه بندی و برچسب های واقعی داده ها به شبیه ترند (به عبارتی الگوی مشابه ای در خوشه بندی و برچسب های داده ها وجود دارد)و این یعنی خوشه بندی بهتری رخ داده است بدون وجود اثر شانس. مقدار ۱ برای این معیار یعنی agreement خوبی اتفاق افتاده بین دو خوشه بندی. مقدار ۰ یعنی هیچ اطلاعات متقابلی وجود ندارد و این یعنی خوشه بندی نسبت به برچسب های وقعی داده ها تقریبا رندوم انجام شده است. مقدار منفی(تا 1-) یعنی خوشه بندی انجام شده نسبت به برچسب واقعی داده ها حتی از رندوم خوشه بندی کردن هم بد تر بوده است.

2-3-2- معیار ARI

این معیار برای اندازه گیری شباهت خوشه بندی حاصل از الگوریتم خوشه بندی و همچنین خوشه بندی حاصل از برچسب واقعی داده ها استفاده می شود. فرمول آن به شکل زیر است:

$$ARI = \frac{(TP + TN) - E[TP + TN]}{\frac{1}{2}(n(n-1)) - E[TP + TN]}$$

برای محاسبه این معیار، به هر جفت از داده ها نگاه می کنیم. اگر هر دو در یک خوشه از الگوریتم خوشه بندی قرار گرفتند، آن جفت را positive و در غیر این صورت آن را negative درنظر می گیریم. حال به وضعیت این جفت داده در برچسب واقعی داده ها نگاه می اندازیم. اگر نتیجه مشابه

با الگوریتم خوشه بندی بود(مثلا دو داده که در یک خوشه از الگوریتم خوشه بندی قرار داشتند، برچسب یکسانی نیز داشتند) پیشوند True و در غیر این صورت پیشوند False قرار می دهیم. حال که این معیار های TN,TP,FP,FN را به دست آوردیم، توسط فرمول بالا این معیار(ARI) را به دست می آوریم. در اینجا هم اثر شانس را توسط کم کردن مقدار مورد انتظار حذف می کنیم(به عبارتی این مقدار موردانتظار نشان می دهد تعداد جفت داده های TP و TN را اگر اختصاص داده ها به خوشه رندوم می بود. این مقدار از توزیع داده ها روی خوشه ها می آید)

در نتیجه با حذف اثر شانس،معیار دقیق تری برای الگوریتم خوشه بندی خود تعریف کرده ایم. این معیار نیز می تواند از ۱ تا منفی ۱ مقدار بگیرد و ۱ نشان دهنده perfect agreement ، صفر نشان دهنده خوشه بندی رندوم و منفی هم نشان دهنده خوشه بندی بدتر از رندوم است.

نکته جالب دیگر درمورد این دو معیار متقارن بودن آنها است. در واقع هر دو agreement بین دو خوشه بندی را اندازه می گیرند(با فرمول های متفاوت) که این مفهوم یک مفهوم متقارن بین دو خوشه بندی است.

نتایج الگوریتم های خوشه بندی و تحلیل آنها

در زیر نیز می توانیم نتایج الگوریتم های خوشه بندی این پروژه را با هم ببینیم:

```
                          Time (s)  Extra Mem (MB)       ARI       AMI  \
Method
NMF                          0.700            0.39  0.612459  0.640784
KMeans-SCRATCH-NMF-REDUCED   0.018            0.00  0.383890  0.537124
KMeans-SCRATCH-NON-REDUCED  10.684            0.29  0.437921  0.531413
KMeans-SKL-NMF-REDUCED       0.006            0.00  0.384601  0.563715
KMeans-SKL-NON-REDUCED       2.302            0.29  0.294220  0.490679

                          Silhouette
Method
NMF                         0.771835
KMeans-SCRATCH-NMF-REDUCED  0.340566
KMeans-SCRATCH-NON-REDUCED  0.015317
KMeans-SKL-NMF-REDUCED      0.454742
KMeans-SKL-NON-REDUCED      0.016199
```

نکاتی که میتوان از این نتایج استنباط کرد:

- از لحاظ زمان، الگوریتم K-means ای که روی ماتریس اصلی tf-idf اعمال شده زمان بیش تری از سایر الگوریتم ها گرفته است که منطقی است. زیرا هم ابعاد ماتریس ورودی بالا است و هم هیچ بهینه سازی در الگوریتم خوشه بندی K-Means اعمال نشده . مثلا در K-Means موجود در SKlearn، ما بهینه سازی هایی مثل vectorization یا موازی سازی انجام می دهیم. مثلا می توانیم عملیات محاسبه فاصله را به شکل موازی انجام دهیم که باعث کاهش زمان الگوریتم می شود. سریع ترین الگوریتم هم مربوط به K-Means SKlearn روی ماتریس W تجزیه نامنفی است. جالب اینجاست الگوریتم NMF هم زمان نسبتا کمی برای اجرا می گیرد. یعنی اگر از نامنفی بودن ماتریس داده خود مطمئن باشیم، با توجه به اینکه الگوریتم K-Means ممکن است در همگرایی کند باشد، روش تجزیه نامنفی برای خوشه بندی، روش مناسبی است.
- از نظر حافظه هم وقتی روی ماتریس W الگوریتم ها را اعمال می کنیم، چون ابعاد آن بسیار کمتر از ماتریس اصلی tf-idf است نیازی به حافظه اضافی نداریم و صرفه جویی مناسبی در حافظه انجام شده است. همچنین روش NMF به خاطر انجام تجزیه ماتریسی و ذخیره ماتریس حاصل W و H، به حافظه بیش تری نیاز دارد.
- از نظر معیار های ARI و AMI ، روش NMF امتیاز بالاتری کسب کرده است. این نشان می دهد این الگوریتم خوشه بندی با شباهت زیاد با برچسب گذاری داده ها انجام می دهد.(معیارARI) همین طور عملکرد kmeans دستی با نسخه Sklearn روی ماتریس W تقریبا مشابه بوده است. این می تواند به این دلیل باشد که با کاهش ابعاد داده، شباهت و ساختار آن نسبت به دیتاست اولیه بر اساس برچسب ها تا حدی از دست رفته است و تفاوت آن چنانی نمی کند از کدام یک استفاده کنیم(از نظر این معیار). در حالت ماتریس tf-idf اصلی، نسخه دستی امتیاز بالاتری کسب کرده است. این می تواند به دلیل تفاوت در خوشه بندی اولیه و همچنین بحث curse of dimensionality باشد. یعنی با

زیاد شدن پیچیدگی دیتا در ابعاد بالا، پیدا کردن الگوهایی برای خوشه بندی سخت تر می شود و همچنین داده sparse تر می شود. حال اگر دو الگوریتم در جزئیات پیاده سازی با هم تفاوت داشته باشند، می تواند به نتایج متفاوتی ختم شود. دقت کنیم همگرایی یکسانی نیز ندارند و آستانه های دو الگوریتم هم برای توقف متفاوت است.

- در معیار AMI، نتایج الگوریتم های kmeans دستی روی ماتریس های W و tf-idf شبیه به هم است و الگوریتم kmeans sklearn روی ماتریس W از بین تمام الگوریتم های kmeans عملکرد بهتری دارد.(معیار AMI)
- معیار silhouetteهم به طور کلی، معیاری است از اینکه یک داده چقدر به خوشه ای که در آن قرار دارد نسبت به خوشه های دیگر شبیه تر است. در واقع معیاری است از کیفیت خوشه بندی و مقادیر 1+ تا 1- را می تواند اختیار کند. هر چه بیش تر باشد، یعنی خوشه ها به خوبی از هم تفکیک شده اند و هر داده بسیار شبیه(نزدیک) به داده های هم خوشه و دور(متفاوت) از داده ها غیر هم خوشه است.
- نتایج بالا نشان می دهد از نظر معیار silhouette الگوریتمNMF امتیاز خوبی کسب می کند و بهترین عملکرد را در مقایسه با دیگر روش ها دارد.همچنین هر دو الگوریتم kmeans دستی و sklearn روی ماتریس اصلی tf-idf امتیاز بسیار پایینی در این معیار دریافت می کنند. نشان می دهد خوشه ها در این حالت به خوبی از هم جدا نیستند و داده ها نزدیک به مرز بین خوشه ها هستند. همچنین روی ماتریس W، الگوریتم KMeans SKlearn خوشه های dense تر و جداپذیر تری را نسبت به الگوریتم kmeans دستی می دهد.

لینک پروژه

https://colab.research.google.com/drive/1MjqWrgZffd04ESDClAcKyeaSlOE1Yuut_-scrollTo=62FWUD1jp0GD