گزارش پروژه سوم داده‌کاوی

عبدالله عبادی ۹۷۴۰۰۰۶۸

۱. مرج داده‌ها

ابتدا داده‌های موجود را که شامل دو فایل movie_info.csv و movie_synopsis.json می‌باشند را باهم مرج کرده و ذخیره می‌کنیم.

```
def merge_data():
    df1 = pd.read_csv("data/movie_info.csv")
    df2 = pd.read_json("data/movie_synopsis.json", orient='index')
    df1.rename(columns={'locale_id': 'local_id'}, inplace=True)
    df_merged = pd.merge(df1, df2, on=['local_id'])
    return df_merged
```

۲. پیش‌پردازش‌ها

این مرحله یکی از مهم‌ترین مراحل و تاثیرگذار بر بخش‌های آتی می‌باشد. ما باید رکوردهای خالی، کلامت اضافه، علائم نگارشی و ... را حذف کنیم. هرچیزی که برای ما ارزش افزوده ندارد باید در این بخش حذف گردد.

```
def pre_process(df):
    df['clean_plot_synopsis'] = df['plot_synopsis'].apply(
        lambda x: remove_stopwords(lemmatize_words(remove_stopwords(remove_punctuations(x)))))
    # stemming
    df['clean_plot_synopsis'] = df['clean_plot_synopsis'].apply(ps.stem)
    print(df.iloc[2]['plot_synopsis'])
    print(df.iloc[2]['clean_plot_synopsis'])
    return df
```

۱) سوال ۱: تفاوت stemming و lemmatization:

این دو روش هردو از روش‌های ریشه‌یابی کلمات هستند با این تفاوت که در stemming کلماتی که در نتیجه حاصل می‌شوند لزوما با معنا نیستند. اما در lemmatization کلمات حاصل با معنا می‌باشند.

۲) تمرین ۱: انجام پیش‌پردازش‌ها و مقایسه نتایج با داده‌های خام.

این بخش شامل چند مرحله است:

- **حذف علائم نگارشی**

در این بخش هرچیزی به جز حروف الفبای انگلیسی را حذف می‌کنیم و تمام حروف را به حروف کوچک تبدیل می‌کنیم.

```
def remove_punctuations(text):
    for punctuation in string.punctuation:
        text = text.replace(punctuation, '')
    text = text.replace(',', ' ')
    text = text.replace('.', " ")
    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]", " ", text)
    # remove whitespaces
    text = ' '.join(list(map(str.strip, text.split())))
    text = text.lower()
    return text
```

- **حذف stop word ها**

stop word ها شامل کلماتی هستند که اطلاعات و ارزش افزوده زیادی برای ما ندارند و ممکن است بارها تکرار شده و تحلیل ما را با خطا روبه‌رو کنند. این کلمات شامل کلماتی مانند: however, it, she, he و ... می‌شوند.

```
def remove_stopwords(text):
    pronouns = ['i', 'he', 'she', 'it', 'him', 'they', 'we', 'us', 'them']
    others = ['tell', 'go', 'take', 'man', 'one', 'back', 'try', 'two', 'time', 's', 'l', 'also', 'become', 'away',
              'next',
              'name', 'call', 'first', 'however', 'john', 'still', 'jack', 'would', 'get', 'see', 'make', 'ask', 'come',
              'day', 'new', 'know', 'later', 'want', 'look', 'help', 'year', 'another', 'like', 'well', 'harry', 'long',
              'visit', 'find', 'leave', 'ed', 'jim', 'le', 'say', 'give', 'mr', 'did', 'take', 'turn', 'end', 'film',
              'set', 'three', 'even', 'several', 'place', 'meanwhile', 'finally', 'soon', 'door', 'talk', 'use',
              'frank',
              'tom', 'continue', 'last', 'together', 'never', 'dr', 'able', 'agree', 'allow', 'begin', 'believe',
              'bring',
              'change', 'belasco', 'muffy', 'leeloo', 'mccabe', 'sub', 'kinjanja', 'hyp', 'chizhov', 'ande',
              'littlefoot',
              'simba', 'boffano', 'helsing', 'renfield', 'nosferatu', 'lestat', 'hutter', 'voysey', 'kotov',
              'pittsburgh',
              'arizona', 'winger', 'bilko', 'robbin', 'hatchett', 'mowgli', 'ripley', 'spock', 'mccoy', 'picard',
              'mcclane',
              'kelso', 'leary', 'jeffrie', 'dc', 'chekhovs', 'miss', 'jos', 'desdemona', 'brighton', 'crockett',
              'simms',
              'deckard', 'minton', 'las', 'vegas', 'dumbo', 'norbu', 'albrecht', 'roseman', 'orlok', 'katherine',
              'indy',
              'mrs', 'gotham', 'barne']

    stop_words = stopwords.words('english') + pronouns + others
    text_with_no_stop_words = [w for w in text.split() if not w in stop_words]
    return ' '.join(text_with_no_stop_words)
```

- **ریشه‌یابی کلمات**

در این بخش در پارگراف‌ها به‌دنبال کلماتی میگردیم که ریشه آنها ساده‌تر است و عوض کردن آن کلمه با ریشه آن برای ما به صرفه‌تر و ساده‌تر است. به عنوان مثال میتوانیم به‌جای killed از kill استفاده کنیم.

```
def lemmatize_words(text):
    words = text.split()
    words = [lemmatizer.lemmatize(word, pos='v') for word in words]
    return ' '.join(words)
```

حال نتیجه پیش پردازش را روی یک پاراگراف می‌بینیم: (خط اول متن اصلی و خط دوم متن پیش پردازش شده است.)

```
The film begins with Ted the Bellhop (Tim Roth) in a room filled with hotel memorabilia, talking to Sam the
ted bellhop tim roth room fill hotel memorabilia sam bellhop marc lawrence bellhop years evethe ingredient
```

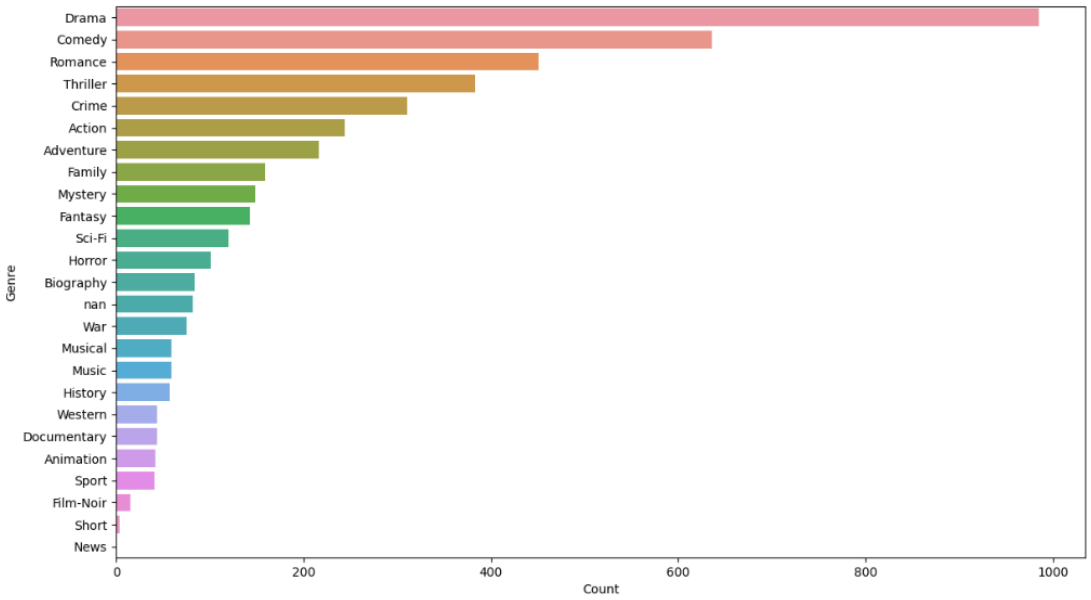
همانطور که مشاهده می‌کنید در متن پیش پردازش شده کلامت اضافی حذف شده است و فقط کلمات اصلی جمله وجود دارد. همچنین علائم نگارشی نیز حذف شده است و در حال حاضر داده‌ها برای انجام تحلیل‌های مختلف بهتر هستند.

۳. استخراج ویژگی

در این مرحله ما به‌دنبال استخراج کلمات پرتکرار و پیدا کردن ژانرهای متفاوتی که در داده‌های این فیلم‌ها وجود دارند هستیم. دی این بخش می‌بینیم که در هر ژانر چه کلماتی بیشتر تکرار شده است.

```python
def calculate_genres_count(df):
    genres_list = []
    for i in df['genre_imdb']:
        genres_list.append(str(i).split('|'))
    genres = sum(genres_list, [])
    len(set(genres))
    genres = nltk.FreqDist(genres)
    genres_count_df = pd.DataFrame({'Genre': list(genres.keys()), 'Count': list(genres.values())})
    g = genres_count_df.nlargest(columns="Count", n=50)
    fig, ax = plt.subplots(figsize=(14, 8))
    ax = sns.barplot(data=g, x="Count", y="Genre")
    plt.show()
    return genres
```

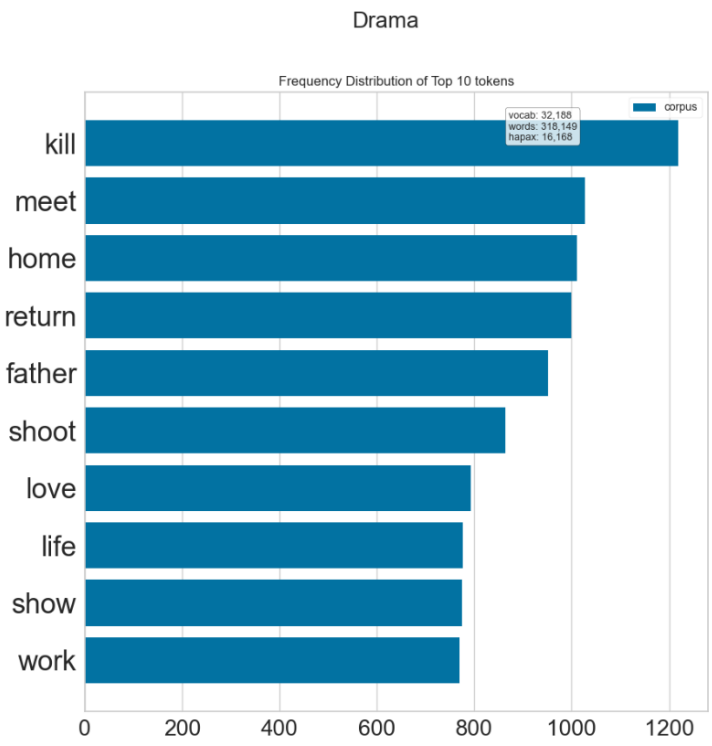
ابتدا در نمودار زیر تعداد تکرار هر ژانر را می‌بینیم. (چه ژانرهایی متداول‌تر هستند)



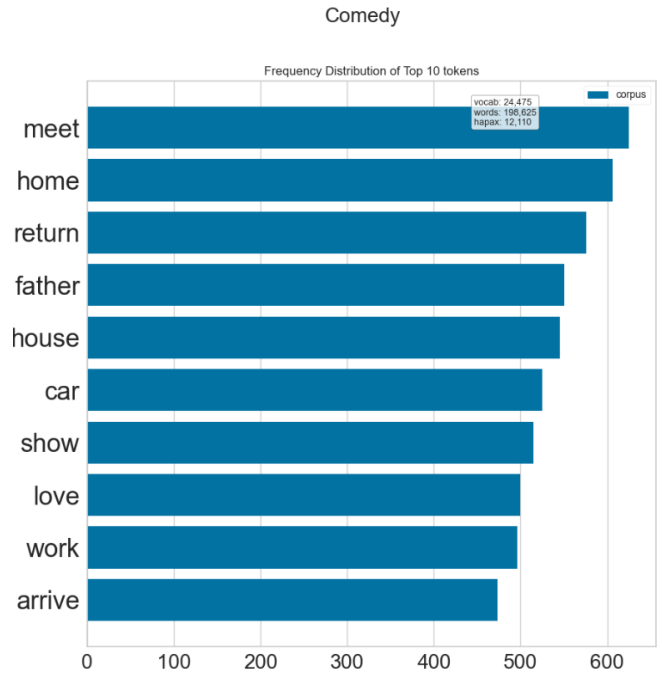
حال کلمات پرتکرار به ازای هر ژانر را پیدا می‌کنیم. در این بخش نمودار کلمات پرتکرار و تعداد تکرار آنها را برای ۴ را ژانر را آورده‌ایم. برا دیدن تمام ژانرها می‌توانید کد را اجرا کنید.

```
def frequent_word_per_genre(df, genre):
    plot = df.loc[df['genre_imdb'].str.contains(genre, na=False), ['clean_plot_synopsis']]
    plotlist = [x for x in plot['clean_plot_synopsis'].str.split()]
    plotlist = list(itertools.chain(*plotlist))
    count = CountVectorizer()
    docs = count.fit_transform(plotlist)
    features = count.get_feature_names_out()
    fig = plt.figure(figsize=(10, 10))
    plt.suptitle(genre, size=20)
    plt.yticks(fontsize=25)
    plt.xticks(fontsize=20)
    visualizer = FreqDistVisualizer(features=features, n=10)
    visualizer.fit(docs)
    visualizer.show()
```

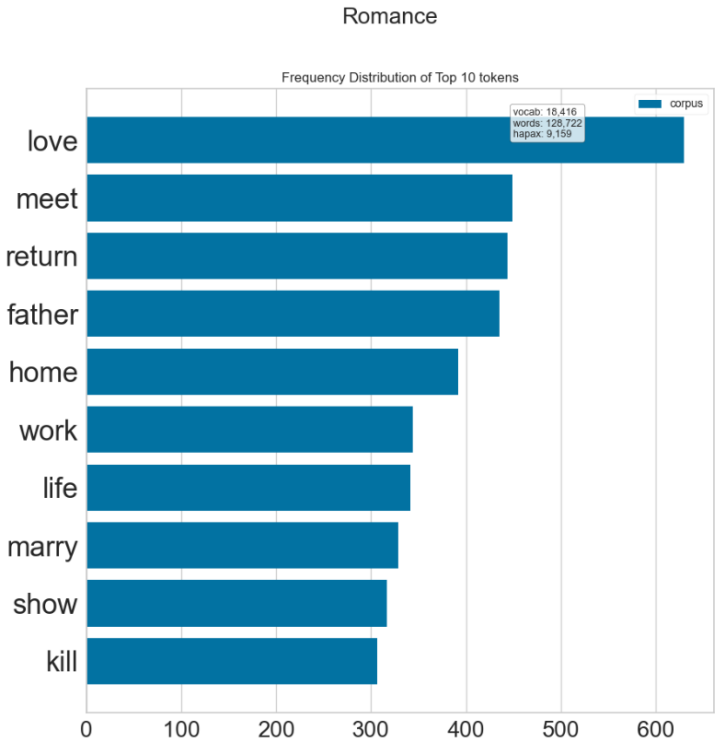
- درام:



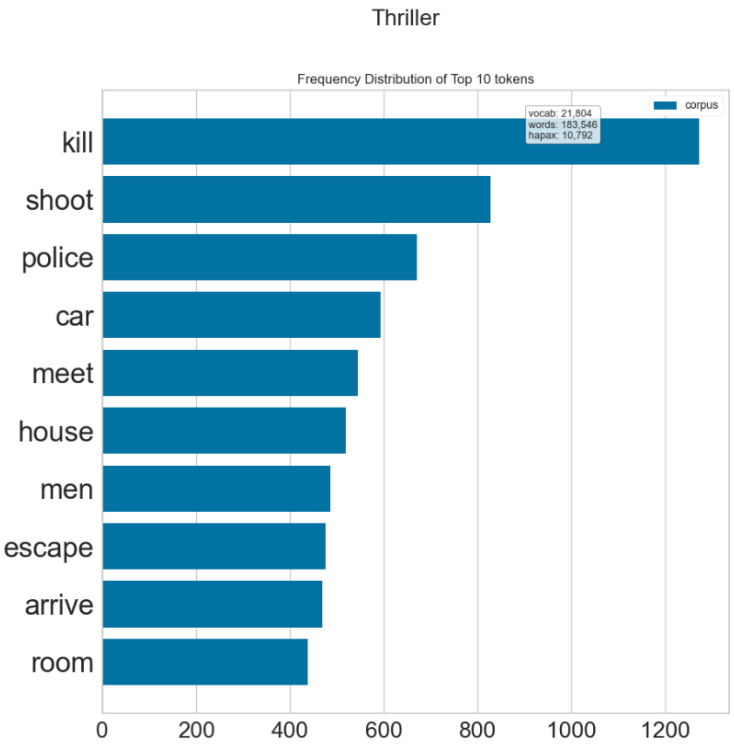
- کمدی:


Comedy
Frequency Distribution of Top 10 tokens
corpus
vocab: 24,475
words: 198,625
hapax: 12,110
meet
home
return
father
house
car
show
love
work
arrive
0
100
200
300
400
500
600

- رمان:


Romance
Frequency Distribution of Top 10 tokens
corpus
vocab: 18,416
words: 128,722
hapax: 9,159
love
meet
return
father
home
work
life
marry
show
kill
0
100
200
300
400
500
600

- هیجانی:



۳) سوال۲: چند نمونه دیگر از روش‌های استخراج ویژگی

- Bag-of-Words :Bag of Words یکی از اساسی ترین روش ها برای تبدیل توکن ها به مجموعه ای از ویژگی ها است. مدل BoW در طبقه بندی اسناد استفاده می شود، جایی که هر کلمه به عنوان یک ویژگی برای آموزش طبقه بندی کننده استفاده می شود. به عنوان مثال، در یک کار تجزیه و تحلیل احساسات مبتنی بر بازبینی، وجود کلماتی مانند "عالی"، "عالی" نشان دهنده یک بررسی مثبت است، در حالی که کلماتی مانند "آزاردهنده"، "ضعیف" به یک بررسی منفی اشاره دارد.
- One hot encoding
- N-grams

۴) تمرین۲: پیاده سازی یک روش . انجام استخراج ویژگی (tf-idf)

در ادامه با روش tf_idf به محاسبه نمره tf_idf می‌پردازیم. TF-IDF که مخفف - Term Frequency Inverse Document Frequency می‌باشد و به معنای فراوانی وزنی کلمه کلیدی است. TF-IDF صرفا میزان تکرار یک کلمه کلیدی یا عبارت را در صفحه نشان نمی‌دهد، بلکه هدف آن نشان دادن اهمیت کلمه کلیدی مورد نظر از طریق مقایسه تعداد تکرار کلمه در متن با تکرار آن کلمه در مجموعه ای بزرگ‌تر از مستندات می‌باشد.

```python
def tfidf(df):
    tfidf_vect = TfidfVectorizer(max_df=0.7, min_df=5, max_features=None, ngram_range=(1, 3))
    plots = df['clean_plot_synopsis'].map(str)
    tf = tfidf_vect.fit_transform(plots)
    words = tfidf_vect.get_feature_names_out()
    words_tfidf_sums = tf.sum(axis=0)
    words_tfidf_scores = []
    output = []
    for i in range(len(words)):
        words_tfidf_scores.append([words[i], words_tfidf_sums[0, i]])
    words_tfidf_scores.sort(key=lambda x: x[1], reverse=True)
    print("words with high tfidf:")
    for i in range(100):
        print('{:<15s}{:>8.2f}'.format(words_tfidf_scores[i][0],
                                       words_tfidf_scores[i][1]))
        output.append([words_tfidf_scores[i][0], words_tfidf_scores[i][1]])
        np.savetxt("tfidf_result_scores.csv", output, delimiter=",", fmt='%s')
    return words_tfidf_scores
```

خروجی این تابع نمرات محاسبه شده برای کلامات به صورت نزولی می‌باشد که در تصویر زیر خروجی را می‌بینید.

```
words with high tfidf:
kill             28.92
father           25.21
home             20.23
meet             20.19
house            20.16
love             19.97
return           19.83
mother           19.80
family           19.07
shoot            19.05
police           18.91
life             17.95
work             17.63
show             16.90
live             16.74
car              16.61
attempt          15.95
run              15.94
```

۴. پردازش داده‌ها:

در این مرحله برای انجام پردازش‌ها و تحلیل‌ها روی داده به‌دنبال یک روش مناسب می‌گردیم و آن را پیاده سازی می‌کنیم. یکی از تحلیل‌های مهم داده‌کاوی که روش‌ها مختفی دارد خوشه‌بندی است و ما یک نمونه از آن را در این بخش پیاده سازی می‌کنیم.

۵) مزایا و معایب خوشه‌بندی و انتخاب یک روش برای این دیتاست

Kmeans یکی از روش‌های معروف خوشه‌بندی است. از مزایای این روش میتوان به سادگی پیاده‌سازی، بازدهی بالا و پیچیدگی کم اشاره کرد. البته این روش معایبی نیز دارد مثلا، از قبل باید مقدار k را به آن بدهیم، همچنین این روش نسبت به داده‌های پرت و نویز حساس است. از آنجایی که ما داده‌ها را پیش‌پردازش کرده و نویز قابل توجهی وجود ندارد و همچنین با توجه به قسمت قبل و تحلیل‌های انتخاب ویژگی که انجام داده و همچنین با کمک elbow می‌توانیم k مناسب را بیابیم پس روش k-means را برای این داده‌ها انتخاب می‌کنیم. همچنین با توجه به اینکه فیلم‌های مختلف ژانرهای مختلف دارند و عبارات به‌خصوصی برای هر ژانر استفاده می‌شود دسته بندی آن با k-means ساده‌تر می‌باشد.

۶) پیاده سازی kmeans

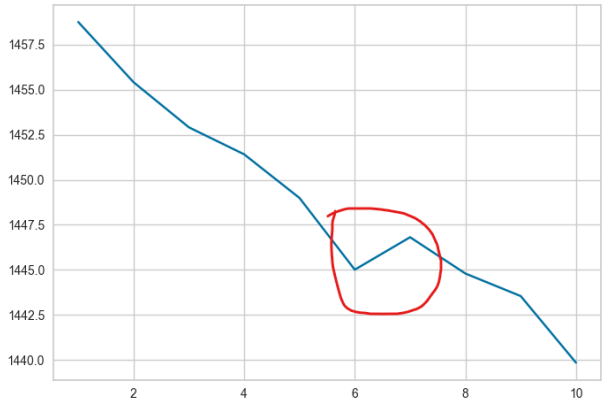
در این بخش به پیاده‌سازی k-means بر روی داده‌های موجود می‌کنیم.

در ابتدا با استفاده از روش elbow و تحلیل‌های انجام شده در بخش قبل سعی می‌کنیم k مناسب برای این داده‌ها را بیابیم.

```
def elbow_k_means(df):
    wcss = []
    for i in range(1, 11):
        clustering = KMeans(n_clusters=i, init='k-means++', random_state=42)
        clustering.fit(df)
        wcss.append(clustering.inertia_)

    ks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    sns.lineplot(x=ks, y=wcss)
    plt.show()
```

خروجی elbow:



همانطور که مشاهده می‌شود تعداد خوشه‌ها می‌تواند بین ۶ و ۸ باشد و با توجه به تحلیل‌های قبل عدد ۸ را انتخاب می‌کنیم.

حال خوشه‌بندی به روش k-means را با k=8 انجام می‌دهیم.

```python
def clustering(df, genres):
    titles = df['title'].tolist()
    synopses = df['clean_plot_synopsis'].tolist()
    # genres = [genres.append(i.split('|')) for i in df['genre_imdb']]
    # define vectorizer parameters
    tfidf_vectorizer = TfidfVectorizer(stop_words='english', use_idf=True)
    tfidf_matrix = tfidf_vectorizer.fit_transform(synopses)  # fit the vectorizer to synopses
    dist = 1 - cosine_similarity(tfidf_matrix)

    # elbow for finding k
    elbow_k_means(tfidf_matrix)

    # KMeans clustering
    num_clusters = 8
    km = KMeans(n_clusters=num_clusters)
    km.fit(tfidf_matrix)
    clusters = km.labels_.tolist()

    # Words in the cluster
    print("words per cluster:")
    order_centroids = km.cluster_centers_.argsort()[:, ::-1]
    terms = tfidf_vectorizer.get_feature_names_out()
    for i in range(num_clusters):
        print("Cluster %d:" % i)
        for ind in order_centroids[i, :5]:
            print(' %s' % terms[ind], end=',')
        print()
```

در خروجی و بعد از انجام خوشه بندی دقت می‌کنیم که هر چه لغاتی در چه خوشه‌هایی بیشتر تکرار شده اند:

```
words per cluster:
Cluster 0:
 moretti, gerardo, lymphoma, insomnia, hodgkins,
Cluster 1:
 kill, father, mother, home, house,
Cluster 2:
 marty, doc, biff, lorraine, george,
Cluster 3:
 lucy, george, joe, beauty, peter,
Cluster 4:
 sally, claw, sandy, christmas, oogie,
Cluster 5:
 kristy, luca, stacey, mary, anne,
Cluster 6:
 beaver, wally, bike, ward, football,
Cluster 7:
 janie, leann, dana, cliftons, russell,
```

۷) بررسی نتایج خوشه‌بندی

باتوجه به نتایج به‌دست‌آمده می‌توان تحلیل‌های متفاوتی با توجه به کلمات و ژانرها انجام داده. مثلا می‌توان از کلمه kill متوجه شد که خوشه یک شامل فیلم‌های اکشن است و باتوجه به کلمات دیگر father, mother, home, house می‌توانم متوجه شد که این کلمات نیز در این فیلم‌ها پر استفاده بوده که بیشتر مربوط به ژانرهای درام و خانوادگی می‌باشد. پس میتوان حدس رد که فیلم‌هایی اکشن این دیتاست ممکن است هم‌زمان ژانر درام نیز باشند و احتمالا فیلم‌هایی هستند که هم‌زمان با اکشن بودن موضوعات درام و خانوادگی نیز در آن جریان دارد.