

Information Retrieval 2021

Instructions for running your system (Engineering a Complete System)

1. Firstly, you need to download Elasticsearch (<https://www.elastic.co/cn/downloads/elasticsearch>), run it in your computer (go to the "bin" directory and run it).

If running successfully, open browser and enter "<http://localhost:9200/>", you will see this:

← → ↻ ⓘ localhost:9200

```
{
  "name" : "LAPTOP-JQLVT4DT",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "83yp9HHhTvuKxp6hgN48YA",
  "version" : {
    "number" : "7.11.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "ff17057114c2199c9c1bbecc727003a907c0db7a",
    "build_date" : "2021-02-15T13:44:09.394032Z",
    "build_snapshot" : false,
    "lucene_version" : "8.7.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

2. Because my program written by Python, you also need the environment for Python.

- "*python -m pip install elasticsearch*"

3. Just run the code, you will this:

```
whole data: shape (34886, 8)
columns: ['Release Year', 'Title', 'Origin/Ethnicity', 'Director', 'Cast', 'Genre', 'Wiki Page', 'Plot']
sample data: shape (1000, 8)
the basic info about those movie collection:

Release Year: [1911, 1912, 1913, 1915, 1917, 1918, 1919, 1920, 1921, 1923, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017]
Origin/Ethnicity: ['Turkish', 'Egyptian', 'Punjabi', 'Tamil', 'Chinese', 'South Korean', 'American', 'Russian', 'Malayalam', 'Telugu', 'Hong Kong', 'Japanese', 'Malaysian', 'Australian', 'Bengali', 'Bollywood', 'British', 'Marathi', 'Filipino', 'Canadian', 'Kannada']
Genre: ['romance', 'clay animation', 'romantic comedy', 'unknown', 'historical film', 'martial arts', 'mythology', 'crime', 'disaster', 'comedy horror', 'devotional', 'world war ii', 'comedy-drama', 'action', 'boxing drama', 'post-apocalyptic science fiction', 'documentary', 'sports comedy', 'sci-fi comedy', 'musical fantasy', 'social', 'crime thriller', 'spy drama', 'adventure', 'spy', 'yakuza', 'family', 'short', 'crime comedy', 'animation', 'science fiction', 'adult comedy', 'drama based on the novel by russell banks', 'war drama', 'action comedy', 'action thriller', 'sci-fi', 'anime for children', 'family', 'drama film', 'comedy', 'propaganda', 'sports', 'drama[not in citation given]', 'musical', 'family drama', 'superhero', 'anime', 'science fantasy', 'western', 'jidaigeki', 'fighting', 'drama', 'romantic drama triangle', 'magical girl', 'neo-noir', 'environmental film', 'slice of life', 'biopic', 'erotic drama', 'anime fantasy', 'thriller', 'epic', 'dark comedy', 'comedy short', 'historical', 'crime drama', 'spy comedy', 'slapstick', 'social thriller', 'mockumentary', 'animated short', 'horror comedy', 'war', 'suspense', 'thriller', 'anime drama mystery fantasy', 'horror thriller', 'mythology (fiction)', 'action', 'action-adventure', 'horror', 'musical comedy', 'dramedy', 'family romance', 'animated', 'romance thriller', 'western comedy', 'action drama', 'dance', 'comedy drama', 'romantic comedy-drama', 'coming-of-age drama', 'anime drama', 'mystery', 'disaster film', 'docudrama', 'biblical', 'erotic thriller', 'wuxia', 'biographical', 'rom com', 'romantic drama', 'biography', 'parody', 'tokusatsu', 'serial', 'bio-pic', 'fantasy', 'film noir', 'family adventure', 'human rights']

indexing documents ...
index name: ir_hw      doc type: movie
upload a sample of 1000 articles with full text to Elasticsearch

Sentence Splitting, Tokenization and Normalization ...
Selecting Keywords ...
Stemming or Morphological Analysis
here is the used analysis filter & analyzer for following search:
{'settings': {'analysis': {'filter': {'english_stop': {'type': 'stop', 'stopwords': '_english_'}, 'light_english_stemmer': {'type': 'stemmer', 'language': 'light_english'}, 'english_possessive_stemmer': {'type': 'stemmer', 'language': 'possessive_english'}}, 'analyzer': {'english': {'tokenizer': 'standard', 'filter': ['english_possessive_stemmer', 'lowercase', 'english_stop', 'light_english_stemmer', 'asciifolding']}}}}}

Searching ...
=====welcome to my IR: =====

Do you want to set the range of Release Year? Y/N:
```

In the screenshot:

It shows the basic statistic information about the sample data (1000 movies):


*“**Release Year, Origin/Ethnicity, Genre**” value enumerations, which will give you some guidance when you want to set those fields.*

4. *Then you can decide the fields values.*

*If you enter “Y”, you start to decide the field values, for example the “**Release Year**” that is a number. And for “**Origin/Ethnicity, Genre**”, they are string/text separated by “,”.*

If you enter “N”, it means anything about this field is ok.

=====welcome to my IR: =====



```
Do you want to set the range of Release Year? Y/N: Y
please enter from which year, for example: 2000. 1990
please enter to which year, for example: 2020. 2010
Do you want to set certain Origin/Ethnicity? Y/N: Y
please enter which origin(s), for example: Hong Kong, American, British...Hong Kong, British, American, Chinese
Do you want to set certain Genre? Y/N: Y
please enter which genre(s), for example: drama, action, comedy, war, romantic...war, comedy, drama
please input your query text: just some movie about war or about family love, or just some comedy movie that is funny
```



find the top 10 most relevant results:

```
index: 12238    relevant score: 9.893341
Release Year: 1994    Origin/Ethnicity: American
Genre: drama
Title: Little Women
Plot: The film focuses on the March sisters: beautiful Meg, tempestuous Jo, tender Beth, and romantic Amy, ...
```

```
index: 20843    relevant score: 9.538972
Release Year: 2004    Origin/Ethnicity: British
Genre: drama
Title: Ae Fond Kiss...
Plot: Set in Glasgow, the film tells the story of the Khan family. Casim is the only son of Pakistani Musl ...
```

```
index: 13558    relevant score: 9.107624
Release Year: 1999    Origin/Ethnicity: American
Genre: drama
Title: The Straight Story
```

5. *According to your entered info, the retrieved results are displayed, if you want to see the detail, just enter the “**index**” number. After that, you can choose to continue searching or quit.*

Do you want to see the detail about the movie? Y/N: Y
please enter the index of the movie(, for example: 621, 11030...): 12238
here is the detail about this article:

1994
Little Women
American

Gillian Armstrong

Winona Ryder, Claire Danes, Trini Alvarado, Kirsten Dunst, Susan Sarandon, Christian Bale, Gabriel Byrne, drama

[https://en.wikipedia.org/wiki/Little_Women_\(1994_film\)](https://en.wikipedia.org/wiki/Little_Women_(1994_film))

The film focuses on the March sisters: beautiful Meg, tempestuous Jo, tender Beth, and romantic Amy, who a Civil War. With their father away fighting in the war, the girls struggle with major and minor problems on Marmee. As a means of escaping some of their problems, the sisters revel in performing in romantic plays w Living next door to the family is wealthy Mr. Laurence, whose grandson Theodore, nicknamed "Laurie", moves arly Jo. Mr. Laurence becomes a mentor for Beth, whose exquisite piano-playing reminds him of his deceased e.

Mr. March is wounded in the war and Marmee is called away to nurse him. While Marmee is away, Beth contrac Meg and Jo send Amy away to live in safety with their Aunt March. Prior to Beth's illness, Jo had been Aun her position she tolerated it in the hope her aunt one day would take her to Europe. When Beth's condition time for Christmas. Mr. Laurence gives his daughter's piano to Beth, Meg accepts John Brooke's proposal an Four years pass; Meg (now twenty) and John marry, and Beth's health is deteriorating steadily. Laurie grad to London with him, but realizing she thinks of him more as an older brother than a lover, she refuses his has decided to take Amy, who is now seventeen, with her to Europe instead of her. Crushed, Jo departs for here she meets Friedrich Bhaer, a German professor who challenges and stimulates her intellectually, intro stories than the lurid Victorian melodramas she has penned so far.

In Europe, Amy is reunited with Laurie. She is disappointed to find he has become dissolute and irresponsi h family. In return, he bitterly rebukes her for courting one of his wealthy college friends in order to m ile he works in London for his grandfather and makes himself worthy of her.

Jo is summoned home to see eighteen year old Beth, who finally dies of the lingering effects of scarlet fe e past four years. A saddened Jo retreats to the comfort of the attic and begins to write her life story. gives birth to fraternal twins Demi and Daisy.

A letter from Amy informs the family that Aunt March is too ill to travel, so Amy must remain in Europe wi rms him of Beth's death and mentions Amy is in Vevey, unable to come home. Laurie immediately travels to b wife, much to Jo's surprise and eventual delight.

Aunt March dies and she leaves Jo her house, which she decides to convert into a school. Professor Bhaer a mistakenly believes Jo has married Laurie he departs to catch a train to the West, where he is to become a he begs him not to leave, he proposes marriage and she happily accepts.

Do you want to continue searching? Y/N: |

Indexing

1. Download the dataset, this link as given in the assignment:

- "https://www.kaggle.com/jrobischon/wikipedia-movie-plots?select=wiki_movie_plots_deduped.csv"

The screenshot shows the Kaggle interface for the 'Wikipedia Movie Plots' dataset. The dataset is described as 'Plot descriptions for ~35,000 movies' and was updated 2 years ago (Version 1) by JustinR. The page includes a search bar, a sidebar with navigation links, and a main content area with a 'Download (30 MB)' button. A red arrow points to the download button.

2. After downloading the dataset, unzip it, read it as pandas dataframe, then sample 1000 rows from it. Every row is a movie with 8 features/columns:

columns: ['Release Year', 'Title', 'Origin/Ethnicity', 'Director', 'Cast', 'Genre', 'Wiki Page', 'Plot']

```

path = "wiki_movie_plots_deduped.csv"
df = pd.read_csv(path)
#print( df.head() )
print( "whole data: shape", df.shape )

# dataframe cols/features
def DFcols(df):
    cols = []
    for col in df:
        cols.append(col)
    #print( cols )
    return cols

cols = DFcols(df)
print( "columns: ", cols )

# sample of 1000 articles, randomly
num = 1000
sample = df.sample(num, random_state=6)
#print( sample.head() )
print( "sample data: shape", sample.shape )


```

3. There is a problem before indexing the sampled 1000 movies, because there are some **NaN** values, so it needs to be solved. Just set those NaN value as empty.

```

# there may be some NaN values in the sample data
sample = sample.replace(np.nan, '', regex=True) # deal with NaN
# upload a sample of 1000 articles with full text to Elasticsearch
index_list = []
for ind, row in sample[:num].iterrows():
    my_doc = row.to_dict()
    es.index(index=my_index, doc_type=my_doc_type, id=ind, body=my_doc) #, ignore=400)
    index_list.append( ind )
#print( index_list[:5] )
print( "upload a sample of 1000 articles with full text to Elasticsearch" )
print()

```



4. Then to index the sampled 1000 movies, set the mapping of every document, because in Elasticsearch, every row/record is taken as document. And set create index and set mapping. It takes some fields value as keyword.

```
# the document is mapping as the following structure
mapping = {
    "properties": {
        "id": { "type": "long" },
        "Release Year": { "type": "keyword" },
        "Title": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        },
        "Origin/Ethnicity": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        },
        "Director": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        },
        "Cast": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        },
        "Genre": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        },
        "Wiki Page": {
            "type": "keyword"
        },
        "Plot": {
            "type": "text", "fields": { "field": { "type": "keyword" } }
        }
    }
}

# create the index with doc type, put the mapping
es.indices.delete(index=my_index, ignore=[400, 404])
es.indices.create(index=my_index, ignore=400)
es.indices.put_mapping(index=my_index, doc_type=my_doc_type, body=mapping, include_type_name = True)
```

Sentence Splitting, Tokenization and Normalization

1. After setting the document mapping, Elasticsearch will automatically index the sampled 1000 movies, but for the input text (or the query), which needs to be processed.
The query is split by "." to several sentences, then split every sentence to multiple terms/words.

```
# remove stopwords and Tokenization
def tokenization(es, inputText, analyzer="english"):
    #analyzer = ['english'] # stop
    res = es.indices.analyze(body={"analyzer" : analyzer, "text" : inputText})
    tokens = []
    for i in res['tokens']:
        #print(i['token'])
        tokens.append( i['token'] )
    return tokens

# Sentence Splitting
def senSplit(es, inputText, analyzer="english"):
    sen_dic = {}
    s_counter = 1
    sentence_delimiter = '.'
    sentences = inputText.split(sentence_delimiter)
    for sentence in sentences:
        sentence = tokenization(es, sentence, analyzer)
        if len(sentence) > 0:
            sen_dic[s_counter] = sentence
            s_counter += 1
    return sen_dic
```

2. It will always work well when the query length is bigger than 200, because when the query is quite short, there is no need to remove the stop-words, process by TF-IDF and select some keywords among them, just split to terms that all are keywords.

```

def retrieve(es, query, origin, genre, yearFrom, yearTo, genreBool=False):
    keys = ""
    if len(query) > 200:
        sen_dic = senSplit(es, query, "stop")
        weights = calWeight(sen_dic)
        keys = selectKeys(weights)
        keys = " ".join(keys)
    else:
        keys = analyzeToStr(es, query, analyzer="standard")
    #print( keys )
    if genreBool:
        querybody = generateQuery(keys, origin, genre, yearFrom, yearTo)
    else:
        querybody = generateQuery(keys, origin=origin, yearFrom=yearFrom, yearTo=yearTo)

    result = searching(es, my_index, querybody)
    return

```

Selecting Keywords

1. After splitting the input text/query as several sentences and every sentence to terms, term frequency (TF) and inverse document frequency (IDF) are calculated for them.
Take every sentence as document here, and every word in sentence (document) as term.
Choose the top terms in every sentence (document) with biggest weight ($= TF * IDF$)
Those chosen terms are considered as keywords.

```

# form word-set from your input text
def termSet(sen_dic):

# calculate the term frequency for every sentence
def termFre(ws, sen):

import math

# IDF, calculate the idf for every word/token
def termIDF(ws, sen_dic):

# calculate the weight for every word in every document/sentence
# sen_dic, dict that includes many sentences split by inputText
def calWeight(sen_dic):

from collections import Counter
# select keywords
def selectKeys(weights, top=10):

```

Stemming or Morphological Analysis

1. Here just write the requirements as JSON (dict type in Python), let the Elasticsearch do word stemming and morphological analysis.

```


setting2 = {
  "settings": {
    "analysis": {
      "filter": {
        "english_stop": {
          "type": "stop",
          "stopwords": "_english_"
        },
        "light_english_stemmer": {
          "type": "stemmer",
          "language": "light_english"
        },
        "english_possessive_stemmer": {
          "type": "stemmer",
          "language": "possessive_english"
        }
      },
      "analyzer": {
        "english": {
          "tokenizer": "standard",
          "filter": [
            "english_possessive_stemmer",
            "lowercase",
            "english_stop",
            "light_english_stemmer",
            "asciifolding"
          ]
        }
      }
    }
  }
}

```

```

# closr first, then add settings, then open
es.indices.close(index=my_index)
es.indices.put_settings(index=my_index, body=setting2 )
es.indices.open(index=my_index)

```



2. To set the “put_settings”, you should close the Elasticsearch first and then open it again after finishing the setting. Or it will output errors.

Searching

1. To use the Elasticsearch as backend to search, your entered query needs to be processed as JSON format, then let the Elasticsearch automatically process it and retrieve results.

```

# generate query to search, given input Text
def generateQuery(queryText, origin="", genre="", yearFrom=1900, yearTo=2022, search):
    if len(queryText) == 0: # return all
        query_body = { "query":{ "match_all":{} } }
        return query_body

    # basic query
    query_body = { "query": { "bool": { "must": [ { "multi_match": { "query": queryText, "fields": ["_source"] } } ] },
        "filter": [ { "range": { "Release Year":{"gt":yearFrom, "lt":yearTo} } } ] }

    # when user decide certain fields such as Origin/Ethnicity, Genre
    if len(origin) > 0:
        query_body["query"]["bool"]["filter"].append( { "match": { 'Origin/Ethnicity': origin } } )
    if len(genre) > 0:
        query_body["query"]["bool"]["filter"].append( { "match": { 'Genre': genre } } )
    return query_body

```

2. After getting the retrieved results that are JSON (dict type in Python), it is necessary to parse the data and show them in free data. Only the top results with biggest relevant scores are displayed.

```

# search, print the top 10 recall results
def searching(es, my_index, querybody):
    result = es.search(index=my_index, body=querybody)
    recallNum = result['took']
    recallContent = result['hits']['hits']
    top = 10
    if top > recallNum:
        top = recallNum
    print("find the top ", top, " most relevant results: \n" )
    for it in recallContent[:top]:
        print("index: ", it['_id'], "\t relevant score: ", it['_score'] )
        content = it['_source']
        print("Release Year: ", content['Release Year'], "\t Origin/Ethnicity: ", content['Origin/Ethnicity'])
        print("Genre: ", content['Genre'])
        print("Title: ", content['Title'])
        print("Plot: ", content['Plot'][:100], "...")
        print()
    return result

```

3. You could also download the Kibana and run it after modifying the “*kibana.yml*” file as below, to visualize the indexed data at “*http://localhost:5601/app/home#*”


```
# Kibana is served by a back end server. This setting specifies the port to use.
server.port: 5601

# Specifies the address to which the Kibana server will bind. IP addresses and hostnames are all valid.
# The default is 'localhost', which usually means remote machines will not be able to connect.
# To allow connections from remote users, set this parameter to a non-loopback address.
server.host: "localhost"

# Enables you to specify a path to mount Kibana at if you are running behind a proxy.
# Use the `server.rewriteBasePath` setting to tell Kibana if it should remove the path
# from requests it receives, and to prevent a deprecation warning at startup.
# This setting cannot end in a slash.
server.basePath: ""

# Specifies whether Kibana should rewrite requests that are prefixed with
# `server.basePath` or require that they are rewritten by your reverse proxy.
# This setting was effectively always `false` before Kibana 6.3 and will
# default to `true` starting in Kibana 7.0.
server.rewriteBasePath: false

# Specifies the public URL at which Kibana is available for end users. If
# `server.basePath` is configured this URL should end with the same basePath.
server.publicBaseUrl: ""

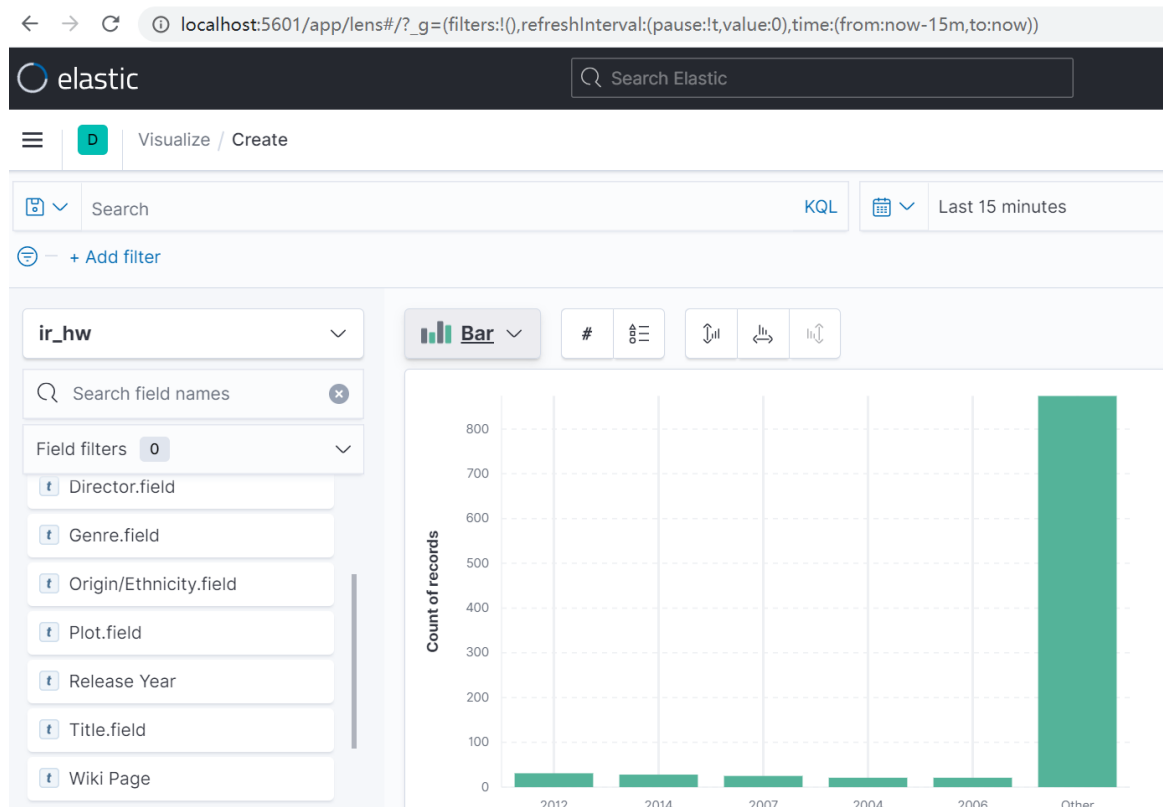
# The maximum payload size in bytes for incoming server requests.
server.maxPayloadBytes: 1048576

# The Kibana server's name. This is used for display purposes.
server.name: "your-hostname"

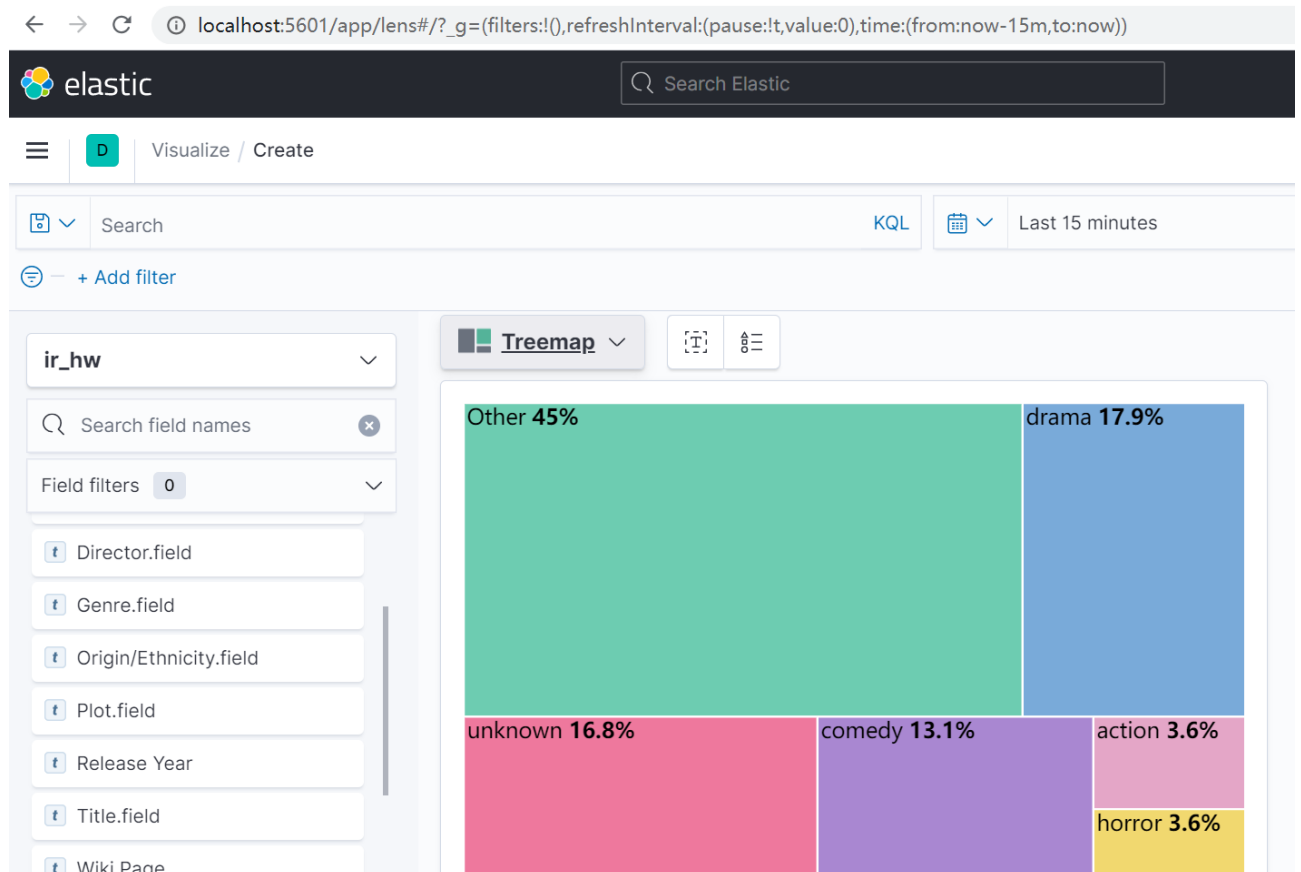
# The URLs of the Elasticsearch instances to use for all your queries.
elasticsearch.hosts: "http://localhost:9200"

# Kibana uses an index in Elasticsearch to store saved searches, visualizations and
```

4. For example,
The “Release Year”:



The “Genre”:



The “Origin/Ethnicity”:

