Σύστημα συστάσεων ταινιών με χρήση αλγορίθμου σημασιολογικής εξόρυξης

Submitted by:

Δημήτριος Τοζακίδης**, 02744**

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Πανεπιστήμιο Θεσσαλίας Βόλος**,** Ελλάδα

2022 - 2023

Περίληψη

Αυτό το αρχείο αποτελεί μια αναφορά ενός συστήματος συστάσεων ταινιών, με βάση τον αλγόριθμο Latent Dirichlet allocation (LDA).

Εισαγωγή

Τα συστήματα συστάσεων χρησιμεύουν παντού στο διαδίκτυο, σε διάφορες πτυχές του, όπως τα κοινωνικά δίκτυα, ιστότοποι βιβλιοθηκών, ιστότοποι προβολής ταινιών και στα συστήματα διαφημίσεων. Υπάρχουν διάφορες προσεγγίσεις για την εφαρμογή συστημάτων συστάσεων. Η LDA είναι μια δημοφιλής τεχνική στο topic modeling. Κυρίως χρησιμοποιείται για συστήματα συστάσεων βιβλιογραφικών εργασιών και βιβλίων. Ο λόγος είναι ότι ο αλγόριθμος αυτός βασίζεται στο πλήθος και την επανάληψη των λέξεων ενός κειμένου, ώστε να προσφέρει ικανοποιητικά αποτελέσματα. Στην αναφορά αυτή σκοπός μας είναι να δείξουμε πώς αυτή η τεχνική-αλγόριθμος μπορεί να εφαρμοστεί στον συγκεκριμένο τομέα και να μελετήσουμε πώς αποδίδει.

Εισαγωγή και καθαρισμός των εγγράφων

Ξεκινάμε βρίσκοντας τα κατάλληλα δεδομένα για το project μας. Το dataframe που επιλέξαμε αποτελείται από 14828 γραμμές και 6 στήλες. Κάνουμε drop την 5η στήλη διότι δεν μας χρησιμεύει. Στην συνέχεια επιλέγουμε ένα sample 200 ταινιών ώστε να χτίσουμε το μοντέλο μας βάση αυτών. Ο καθαρισμός δεδομένων είναι απολύτως κρίσιμος για τη δημιουργία ενός χρήσιμου μοντέλου. Τα δεδομένα που μας ενδιαφέρουν για την δημιουργία του μοντέλου είναι η σύνοψη της κάθε ταινίας που χρησιμοποιούμε. Τα τρία παρακάτω βήματα είναι κοινά στις περισσότερες μεθόδους επεξεργασίας φυσικής γλώσσας:

• Tokenizing

• Stopping

• Stemming

Το tokenization τμηματοποιεί ένα έγγραφο στα ατομικά του στοιχεία. Σε αυτή την περίπτωση, μας ενδιαφέρει κάθε λέξη να αριθμηθεί. Το tokenization μπορεί να πραγματοποιηθεί με πολλούς τρόπους. Εμείς με την χρήση του nltk.tokenize.simple αποφεύ-

γουμε προβλήματα όπως το “don’t” να διαβαστεί ως δύο tokens “don” και “t”. Ορισμένα μέρη της αγγλικής ομιλίας, όπως οι σύνδεσμοι “for”, “or” ή η λέξη “the” δεν έχουν νόημα για ένα θέμα του μοντέλου μας. Αυτοί οι όροι ονομάζονται stop words και πρέπει να αφαιρεθούν από τη λίστα των tokens μας. Στην περίπτωσή μας, χρησιμοποιούμε το πακέτο stopwords από την Pypi, μια σχετικά συντηρητική λίστα. Μπορούμε να καλέσουμε τη getstopwords() για να δημιουργήσουμε μια λίστα με stop words. Οι Stemming words βασίζονται στην τεχνική NLP για τη μείωση τοπικά όμοιων λέξεων στη ρίζα τους. Για παράδειγμα, “stemming,” “stemmer,” “stemmed,” όλα έχουν παρόμοιες σημασίες. Το stemming μειώνει αυτούς τους όρους σε ”stem”. Αυτό είναι σημαντικό για τη μοντελοποίηση θεμάτων, η οποία διαφορετικά θα έβλεπε αυτούς τους όρους ως ξεχωριστές οντότητες και θα μείωνε τη σημασία τους στο μοντέλο. Ο αλγόριθμος Porter stemming είναι η πιο ευρέως χρησιμοποιούμενη μέθοδος για αυτό και την επιλέγουμε.

```
In [120]: 1 movies_clean
```

	imdb_id	title	plot_synopsis	tags	synopsis_source
1305	tt0964587	St. Trinian's	Annabelle Fritton (Talulah Riley), an uptight ...	comedy	wikipedia
3263	tt0238546	Queen of the Damned	The vampire Lestat is awakened from decades of...	paranormal, revenge, gothic, murder, flashback	wikipedia
12892	tt0378918	True Crime: Streets of LA	Note: This plot synopsis details the "good end...	revenge, murder, violence	wikipedia
10498	tt1259520	Bon appétit	Daniel (Unax Ugalde) is a young and ambitious ...	romantic	wikipedia
9008	tt0297162	Half Past Dead	In San Francisco, Sasha Petrosevitch (Steven S...	violence, comedy, murder, flashback	wikipedia
...	...	...	...	...	...
5412	tt2359024	Blue Ruin	Dwight (Macon Blair) is a beach vagrant, with ...	revenge, suspenseful, neo noir, violence	imdb
7600	tt0332379	The School of Rock	No Vacancy, a rock band, performs at a nightcl...	cult, entertaining	wikipedia
14530	tt0058921	Arzoo	Gopal (Rajendra Kumar) is a skiing champion. H...	romantic	wikipedia
9913	tt0037096	Mrs. Parkington	At Christmastime in 1938, Susie Parkington, an...	revenge, flashback	wikipedia
13853	tt0098436	The Tall Guy	The protagonist and narrator is Dexter King (G...	romantic, satire, flashback	wikipedia

200 rows × 5 columns

Figure 1: Το ”καθαρισμένο” dataframe

Δημιουργία του πίνακα

Το αποτέλεσμα του σταδίου καθαρισμού μας είναι μια tokenized, stopped και stemmed λίστα λέξεων από ένα μόνο έγγραφο. Στην συνέχεια εφαρμόζουμε αυτή την μέθοδο σε κάθε σύνοψη από τις ταινίες που χρησιμοποιούμε. Για να δημιουργήσουμε ένα μοντέλο LDA, πρέπει να κατανοήσουμε πόσο συχνά εμφανίζεται κάθε όρος σε κάθε έγγραφο. Για να γίνει αυτό, πρέπει να κατασκευάσουμε ένα εγγράφο με ένα πακέτο που ονομάζεται gensim. Η συνάρτηση Dictionary() διασχίζει το έγγραφο εκχωρώντας ένα μοναδικό ακέραιο σε κάθε μοναδικό token, ενώ παράλληλα συλλέγει πλήθος λέξεων και σχετικά

στατιστικά στοιχεία.

```
In [64]: 1 print(dictionary.token2id)
```


```
{'19th': 0, 'abandon': 1, 'abbi': 2, 'absenc': 3, 'acquisit': 4, 'actual': 5, 'agre': 6, 'aid': 7, 'air': 8, 'alfonsi':
9, 'ali': 10, 'alreadi': 11, 'also': 12, 'anderson': 13, 'anticip': 14, 'apart': 15, 'appar': 16, 'appear': 17, 'arm': 1
8, 'around': 19, 'arriv': 20, 'ask': 21, 'assail': 22, 'attack': 23, 'attempt': 24, 'attract': 25, 'await': 26, 'awaken':
27, 'away': 28, 'back': 29, 'basement': 30, 'battl': 31, 'bed': 32, 'beg': 33, 'begin': 34, 'behead': 35, 'behind': 36,
'believ': 37, 'belong': 38, 'beset': 39, 'best': 40, 'bit': 41, 'bite': 42, 'blade': 43, 'blood': 44, 'bode': 45, 'bodi':
46, 'bori': 47, 'branch': 48, 'break': 49, 'breakup': 50, 'brother': 51, 'bruis': 52, 'buri': 53, 'burial': 54, 'busi': 5
5, 'butcher': 56, 'buzz': 57, 'cadav': 58, 'call': 59, 'caller': 60, 'calm': 61, 'camera': 62, 'can': 63, 'care': 64, 'ca
se': 65, 'cathedr': 66, 'caus': 67, 'centuri': 68, 'certain': 69, 'chang': 70, 'chester': 71, 'child': 72, 'close': 73,
'come': 74, 'commit': 75, 'concierg': 76, 'conclud': 77, 'confess': 78, 'confirm': 79, 'confus': 80, 'continu': 81, 'corp
s': 82, 'cottag': 83, 'count': 84, 'cours': 85, 'creep': 86, 'crew': 87, 'crewmen': 88, 'curs': 89, 'd': 90, 'dagger': 9
1, 'daughter': 92, 'dawn': 93, 'day': 94, 'de': 95, 'dead': 96, 'death': 97, 'decid': 98, 'decis': 99, 'demeanor': 100,
'dialina': 101, 'diamond': 102, 'die': 103, 'difficult': 104, 'discov': 105, 'distinct': 106, 'distract': 107, 'distres
s': 108, 'doctor': 109, 'door': 110, 'doubt': 111, 'drain': 112, 'dread': 113, 'dress': 114, 'drip': 115, 'drop': 116, 'd
uti': 117, 'eagerli': 118, 'earlier': 119, 'east': 120, 'elabor': 121, 'elderli': 122, 'embrac': 123, 'end': 124, 'englan
d': 125, 'entranc': 126, 'escap': 127, 'estrang': 128, 'even': 129, 'event': 130, 'ex': 131, 'examin': 132, 'explain': 13
3, 'face': 134, 'fail': 135, 'fake': 136, 'famili': 137, 'father': 138, 'fear': 139, 'feed': 140, 'felt': 141, 'final': 1
42, 'find': 143, 'finger': 144, 'finish': 145, 'fit': 146, 'five': 147, 'flat': 148, 'flee': 149, 'fli': 150, 'floor': 15
1, 'follow': 152, 'forc': 153, 'forest': 154, 'forgiv': 155, 'former': 156, 'frank': 157, 'friend': 158, 'fright': 159,
'front': 160, 'gasp': 161, 'get': 162, 'ghost': 163, 'giorgio': 164, 'girl': 165, 'give': 166, 'glass': 167, 'glauco': 16
```


Figure 2: Το "λεξικο" με τις λέξεις μας

Η συνάρτηση doc2bow() μετατρέπει το Dictionary() σε τσάντα με λέξεις. Το αποτέλεσμα, corpus, είναι μια λίστα vectors ίση με τον αριθμό των εγγράφων. Σε κάθε vector εγγράφου υπάρχει μια σειρά από πλειάδες. Για παράδειγμα, το print(corpus[0]) έχει ως αποτέλεσμα τα εξής:

```
In [66]: 1 print(corpus[0])
```


```
[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 2), (7, 2), (8, 1), (9, 1), (10, 2), (11, 1), (12, 2), (13, 1), (14, 1), (15, 4),
(16, 1), (17, 3), (18, 1), (19, 1), (20, 3), (21, 1), (22, 2), (23, 1), (24, 1), (25, 2), (26, 1), (27, 3), (28, 1), (29, 2), (30, 1),
(31, 1), (32, 5), (33, 4), (34, 2), (35, 2), (36, 1), (37, 1), (38, 1), (39, 1), (40, 1), (41, 1), (42, 2), (43, 1), (44, 3), (45, 1),
(46, 4), (47, 3), (48, 1), (49, 2), (50, 1), (51, 2), (52, 1), (53, 1), (54, 1), (55, 1), (56, 1), (57, 1), (58, 1), (59, 8), (60, 2),
(61, 1), (62, 1), (63, 1), (64, 2), (65, 1), (66, 1), (67, 1), (68, 1), (69, 3), (70, 1), (71, 6), (72, 2), (73, 1), (74, 3), (75, 1),
(76, 2), (77, 2), (78, 1), (79, 1), (80, 1), (81, 2), (82, 5), (83, 5), (84, 2), (85, 1), (86, 1), (87, 1), (88, 1), (89, 1), (90, 1),
(91, 1), (92, 1), (93, 1), (94, 1), (95, 1), (96, 1), (97, 2), (98, 1), (99, 1), (100, 1), (101, 1), (102, 1), (103, 1), (104, 1), (10
5, 3), (106, 1), (107, 1), (108, 1), (109, 1), (110, 1), (111, 1), (112, 1), (113, 1), (114, 1), (115, 1), (116, 3), (117, 1), (118,
1), (119, 1), (120, 1), (121, 1), (122, 1), (123, 1), (124, 1), (125, 1), (126, 1), (127, 2), (128, 1), (129, 2), (130, 1), (131, 1),
(132, 1), (133, 3), (134, 2), (135, 1), (136, 1), (137, 7), (138, 3), (139, 4), (140, 2), (141, 1), (142, 1), (143, 2), (144, 3), (145,
1), (146, 1), (147, 1), (148, 1), (149, 2), (150, 3), (151, 1), (152, 1), (153, 1), (154, 1), (155, 1), (156, 1), (157, 6), (158, 1),
(159, 2), (160, 1), (161, 1), (162, 2), (163, 1), (164, 6), (165, 1), (166, 1), (167, 1), (168, 1), (169, 2), (170, 1), (171, 7), (172,
1), (173, 1), (174, 1), (175, 1), (176, 1), (177, 1), (178, 1), (179, 1), (180, 1), (181, 1), (182, 1), (183, 1), (184, 2), (185, 1),
(186, 1), (187, 4), (188, 1), (189, 2), (190, 1), (191, 1), (192, 1), (193, 1), (194, 1), (195, 2), (196, 1), (197, 1), (198, 3), (199,
2), (200, 1), (201, 1), (202, 1), (203, 1), (204, 3), (205, 1), (206, 1), (207, 1), (208, 1), (209, 1), (210, 3), (211, 2), (212, 5),
(213, 1), (214, 1), (215, 1), (216, 1), (217, 4), (218, 1), (219, 1), (220, 1), (221, 1), (222, 1), (223, 3), (224, 2), (225, 1), (226,
1), (227, 1), (228, 1), (229, 2), (230, 1), (231, 1), (232, 2), (233, 3), (234, 1), (235, 1), (236, 1), (237, 1), (238, 5), (239, 3),
(240, 6), (241, 1), (242, 1), (243, 1), (244, 1), (245, 1), (246, 2), (247, 1), (248, 1), (249, 1), (250, 1), (251, 1), (252, 1), (253,
1), (254, 1), (255, 1), (256, 1), (257, 1), (258, 1), (259, 1), (260, 1), (261, 7), (262, 2), (263, 3), (264, 3), (265, 6), (266, 1),
(267, 1), (268, 1), (269, 1), (270, 3), (271, 5), (272, 1), (273, 1), (274, 1), (275, 1), (276, 1), (277, 1), (278, 1), (279, 1), (280,
1), (281, 1), (282, 3), (283, 1), (284, 4), (285, 1), (286, 2), (287, 1), (288, 1), (289, 1), (290, 1), (291, 1), (292, 1), (293, 1),
(294, 1), (295, 2), (296, 1), (297, 1), (298, 2), (299, 1), (300, 1), (301, 1), (302, 1), (303, 1), (304, 1), (305, 1), (306, 1), (307,
3), (308, 1), (309, 1), (310, 1), (311, 1), (312, 1), (313, 1), (314, 1), (315, 1), (316, 6), (317, 1), (318, 1), (319, 1), (320, 2),
(321, 1), (322, 1), (323, 3), (324, 1), (325, 1), (326, 14), (327, 1), (328, 2), (329, 1), (330, 1), (331, 1), (332, 11), (333, 3), (33
4, 1), (335, 7), (336, 1), (337, 1), (338, 1), (339, 2), (340, 1), (341, 1), (342, 1), (343, 1), (344, 1), (345, 1), (346, 1), (347,
1), (348, 1), (349, 1), (350, 1), (351, 2), (352, 1), (353, 3), (354, 1), (355, 1), (356, 2), (357, 2), (358, 3), (359, 1), (360, 1),
(361, 1), (362, 1), (363, 1), (364, 2), (365, 1), (366, 2), (367, 1), (368, 1), (369, 2), (370, 3), (371, 3), (372, 1), (373, 1), (374,
1), (375, 1), (376, 2), (377, 1), (378, 1), (379, 1), (380, 1), (381, 1), (382, 1), (383, 1), (384, 1), (385, 1), (386, 2), (387, 2),
(388, 2), (389, 1), (390, 1), (391, 1), (392, 1), (393, 2), (394, 1), (395, 1), (396, 3), (397, 1), (398, 1), (399, 2), (400, 1), (401,
1), (402, 1), (403, 1), (404, 1), (405, 1), (406, 1), (407, 3), (408, 1), (409, 1), (410, 1), (411, 1), (412, 1), (413, 1), (414, 1),
(415, 1), (416, 1), (417, 1), (418, 1), (419, 1), (420, 10), (421, 2), (422, 1), (423, 1), (424, 2), (425, 1), (426, 2), (427, 1), (42
8, 1), (429, 2), (430, 1), (431, 1), (432, 1), (433, 4), (434, 1), (435, 1), (436, 1), (437, 1), (438, 4), (439, 1), (440, 4), (441,
1)]
```


Figure 3: Το corpus

Εφαρμόζοντας το **LDA** μοντέλο

Το corpus είναι ένας πίνακας εγγράφων και τώρα είμαστε έτοιμοι να δημιουργήσουμε ένα μοντέλο LDA. Η κλάση LdaModel ανήκει στο gensim και οι παράμετροι που χρησιμοποιούμε είναι οι εξής:

• numtopics: Απαιτείται. Ένα μοντέλο LDA απαιτεί από τον χρήστη να καθορίσει πόσα θέματα πρέπει να δημιουργηθούν. Μετά από δοκιμές επιλέξαμε να ζητήσουμε 30 θέματα

• id2word: Απαιτείται. Η κλάση LdaModel απαιτεί από το προηγούμενο dictionary μας να αντιστοιχίσει τους μοναδικούς ακεραίους σε συμβολοσειρές.

• passes: Προαιρετικός. Ο αριθμός των επαναλήψεων που θα κάνει το μοντέλο μέσω του corpus. Όσο μεγαλύτερος είναι ο αριθμός των περασμάτων, τόσο πιο ακριβές θα είναι το μοντέλο. Πολλά περάσματα μπορεί να είναι αργά σε ένα πολύ μεγάλο corpus.

Εδώ εκτυπώνουμε και τα 30 θέματα. Κάθε καταχώριση θα έχει τις 10 κορυφαίες λέξεις κατά σημαντικότητα, ακολουθούμενες από τη συσχέτιση του συνόλου των θεμάτων, τον μέσο όρο των βαθμολογιών κατά ζεύγη ομοιότητας λέξεων όλων των λέξεων εντός του θέματος.

```
In [121]:
from pprint import pprint
top_topics = lda_model_final.top_topics(corpus, topn=10)

In [122]:
counter = 0
for topic in top_topics:
    print('Topic {}:'.format(counter))
    counter += 1
    pprint(topic)
```

```
Topic 0:
([(0.05555626, 'harrer'),
  (0.024083028, 'aufschnait'),
  (0.01983849, 'chines'),
  (0.01945063, 'lama'),
  (0.018528925, 'tibetan'),
  (0.016672892, 'dalai'),
  (0.015748698, 'tibet'),
  (0.012974823, 'lhasa'),
  (0.010197772, 'jigm'),
  (0.00754309, 'gift')],
 -0.4684910551240771)
Topic 1:
([(0.02063212, 'terri'),
  (0.01857233, 'ian'),
  (0.010655121, 'howard'),
  (0.010333157, 'brother'),
  (0.008542705, 'kill'),
  (0.008378023, 'burn'),
```

Figure 4: Τα 30 θέματα

Ας δούμε τώρα από ποια θέματα αποτελείται η ταινία St. Trinian's που είναι η πρώτη ταινία στο dataframe των 200 ταινιών:

Το υψηλότερο συσχετιζόμενο θέμα ήταν το 11, το οποίο είναι: 'oak', 'telli', 'calvess', 'darnel', 't', 'gun', 'gump', 'say', 'ask', 'talk'.

```
In [154]:
corpus_lda_model = lda_model_final[corpus]
for article in corpus_lda_model[0]:
    print(article)
```

```
(11, 0.80746126)
(22, 0.19123007)
```

```
In [155]:
movies_clean
```

Out[155]:

	imdb_id	title	plot_synopsis	tags	synopsis_source
1305	tt0964587	St. Trinian's	Annabelle Fritton (Talulah Riley), an uptight ...	comedy	wikipedia
3263	tt0238546	Queen of the Damned	The vampire Lestat is awakened from decades of...	paranormal, revenge, gothic, murder, flashback	wikipedia
12892	tt0378918	True Crime: Streets of LA	Note: This plot synopsis details the "good end...	revenge, murder, violence	wikipedia
10498	tt1259520	Bon appétit	Daniel (Unax Ugalde) is a young and ambitious ...	romantic	wikipedia
9008	tt0297162	Half Past Dead	In San Francisco, Sasha Petrosevitch (Steven S...	violence, comedy, murder, flashback	wikipedia
...	...	...	...	...	...
5412	tt2359024	Blue Ruin	Dwight (Macon Blair) is a beach vagrant, with ...	revenge, suspenseful, neo noir, violence	imdb
7600	tt0332379	The School of Rock	No Vacancy, a rock band, performs at a nightcl...	cult, entertaining	wikipedia
14530	tt0058921	Arzoo	Gopal (Rajendra Kumar) is a skiing champion. H...	romantic	wikipedia
9913	tt0037096	Mrs. Parkington	At Christmastime in 1938, Susie Parkington, an...	revenge, flashback	wikipedia
13853	tt0098436	The Tall Guy	The protagonist and narrator is Dexter King (G...	romantic, satire, flashback	wikipedia

200 rows × 5 columns

Figure 5: Τα θέματα της ταινίας St. Trinian's

Ας κάνουμε έναν τελικό έλεγχο με βάση την σειρά των εγγράφων μας κατά μήκος. Θα ήταν λογικό τα μεγαλύτερα άρθρα να αποτελούνται από περισσότερα θέματα από τα μικρότερα, οπότε ας γράψουμε τη θεματική συσχέτιση της οπτικής με την υψηλότερη συσχέτιση για κάθε άρθρο και ας δούμε αν μπορούνε να έχουνε μια φθίνουσα σχέση.

In [149]:
```
from operator import itemgetter
index_high_corr_list = []

index_counter = 0

for article in corpus_lda_model:
    highest_feature = max(article, key=itemgetter(1))[0]
    max_correlation = max(article, key=itemgetter(1))[1]
    index_high_corr_list.append([index_counter, highest_feature, max_correlation])
    index_counter += 1

index_high_corr_list[:5]
```

Out[149]:
```
[[0, 29, 0.55548346],
 [1, 25, 0.99485016],
 [2, 34, 0.45816225],
 [3, 55, 0.9052063],
 [4, 30, 0.9524169]]
```

In [135]:
```
import matplotlib.pyplot as plt
x = [tup[0] for tup in index_high_corr_list]
y = [tup[2] for tup in index_high_corr_list]

plt.figure(figsize=(20,10))
scatter = plt.plot(x,y, '.')
line = plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)), 'r')
plt.title('Highest Correlation of Topics/Movie Length', fontsize=30)
plt.xlabel("Movie Length Rank Ordered", fontsize=20)
plt.ylabel("Highest Topic Correlation", fontsize=20)
plt.show(scatter, line)
```

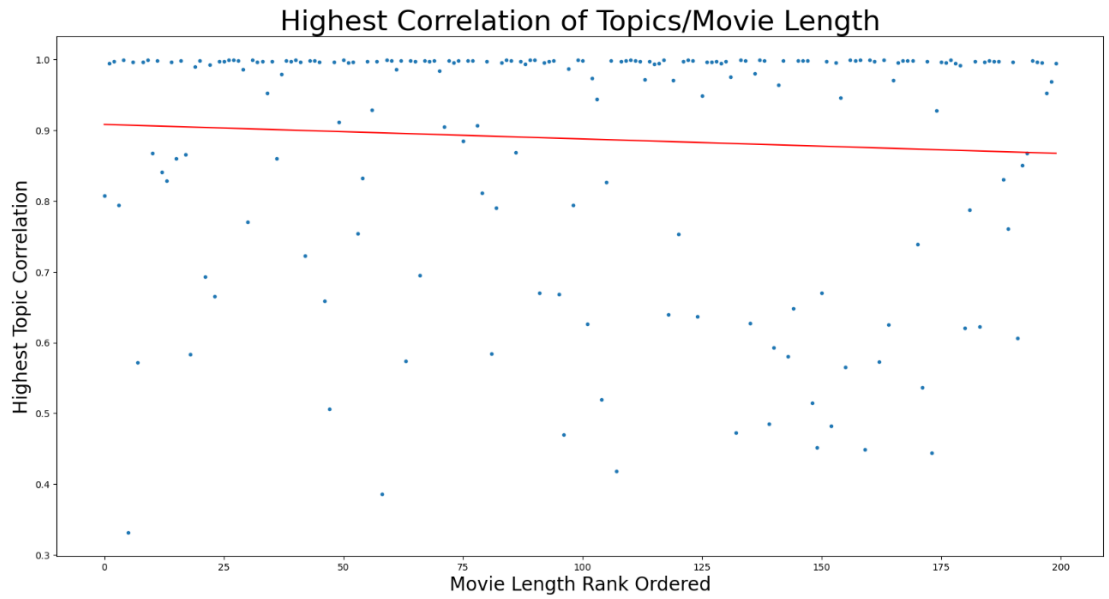

Figure 6: Βλέπουμε όντως οτι ακολουθεί πτητική πορεία

Το σύστημα συστάσεων

Είμαστε πλέον έτοιμοι να κάνουμε τις συστάσεις μας. Θα χρησιμοποιήσουμε την ομοιότητα συνημίτονου για να συγκρίνουμε τα vectors των θεμάτων και της συσχέτισης τους ανά έγγραφο. Η ομοιότητα συνημίτονου μεταξύ των vectors του σάκου λέξεων μας δίνουν ομοιότητες στην περιοχή από -1 έως 1 (όσο μεγαλύτερη, τόσο πιο παρόμοια)

$$\text{Similarity} = \cos(\theta) = (A \cdot B)/(\|A\|\|B\|), \|X\|$$

Ζητούμε αυτά τα αποτελέσματα για συστάσεις με βάση μια ταινία που έχει επιλέξει ο χρήστης. Αρχικοποιούμε τον πίνακα ομοιότητας μεταξύ όλων των εγγράφων, από τον οποίο θα ζητήσουμε για τις απαιτούμενες ταινίες και θα χρησιμοποιήσουμε ως βάση για τις συστάσεις μας. Πραγματοποιούμε τα ακόλουθα βήματα:

• Ένας τίτλος ταινίας εισάγεται από τον χρήστη

• Ο αντίστοιχος τίτλος της ταινίας βρίσκεται στον corpus

• Οι ομοιότητες όλων των άλλων ταινιών αναζητούνται από το ευρετήριο, τον πίνακα ομοιοτήτων μας

• Αυτές οι ομοιότητες απαριθμούνται έτσι ώστε μια βαθμολογία ομοιότητας να μπορεί να αντιστοιχιστεί στον αντίστοιχο τίτλο

• Οι βαθμολογίες προτάσεων για όλες τις ταινίες εισάγονται σε μια κενή λίστα που δημιουργήθηκε προηγουμένως

• Αυτές οι βαθμολογίες συστάσεων είναι ταξινομημένες

• Τα ακόλουθα εκτυπώνονται:

    • Τα πρώτα δέκα πιο συνηθισμένα tokens στο έγγραφο της ταινίας

    • Το σημαντικότερο θέμα της ταινίας

    • Η 2η έως την 11η συστάσεις (το πρώτο στοιχείο είναι η ίδια ταινία που ζητήθηκε)

In [137]:
```python
def movie_recommender(title):
    movies_checked = 0
    for i in range(len(movies_clean)):
        recommendation_scores = []
        if movies_clean.iloc[i][1] == title:
            lda_vectors = corpus_lda_model[i]
            sims = index[lda_vectors]
            sims = enumerate(sims)
            for sim in sims:
                movies_num = sim[0]
                recommendation_score = [movies_clean.iloc[movies_num][1], sim[1]]
                recommendation_scores.append(recommendation_score)

            recommendation = sorted(recommendation_scores, key=lambda x: x[1], reverse=True)
            print("Your book's most prominent tokens are:")
            article_tokens = corpus[i]
            sorted_tokens = sorted(article_tokens, key=lambda x: x[1], reverse=True)
            sorted_tokens_10 = sorted_tokens[:10]
            for i in range(len(sorted_tokens_10)):
                print("Word {} (\"{}\") appears {} time(s).".format(sorted_tokens_10[i][0],
                                                            dictionary[sorted_tokens_10[i][0]],
                                                            sorted_tokens_10[i][1]))
            print('-----')
            print("Your book's most prominant topic is:")
            print(lda_model_final.print_topic(max(lda_vectors, key=lambda item: item[1])[0]))
            print('-----')
            print('Here are your recommendations for "{}":'.format(title))
            display(recommendation[1:11])

        else:
            movies_checked +=1

        if movies_checked == len(movies_clean):
            movie_suggestions = []
            print('Sorry, but it looks like "{}" is not available.'.format(title))
            other_books = []
```

In [138]:
```python
movie_recommender("Pompeii")
```

```
Your book's most prominent tokens are:
Word 1717 ("jim") appears 90 time(s).
Word 1805 ("gang") appears 33 time(s).
Word 332 ("s") appears 30 time(s).
Word 247 ("member") appears 25 time(s).
Word 6301 ("wayn") appears 25 time(s).
Word 143 ("find") appears 18 time(s).
Word 1072 ("student") appears 17 time(s).
Word 162 ("get") appears 16 time(s).
Word 29 ("back") appears 15 time(s).
Word 1848 ("train") appears 15 time(s).
-----
Your book's most prominant topic is:
0.029*"jim" + 0.020*"s" + 0.012*"gang" + 0.010*"simon" + 0.008*"member" + 0.008*"find" + 0.008*"wayn" + 0.007*"mcclane" +
0.007*"get" + 0.007*"kate"
-----
Here are your recommendations for "Pompeii":

[['Red Riding Hood', 0.99961674],
 ['Charly', 0.99961674],
 ['Constantine', 0.99961674],
 ['13 Seconds', 0.99961674],
 ['Eye in the Sky', 0.7193544],
 ['Three Days of the Condor', 0.17535248],
 ['The Cable Guy', 0.02768309],
 ['Kolja', 0.02768309],
 ['Kingdom Hearts: Chain of Memories', 0.02768309],
 ['Attack of the Crab Monsters', 0.02768309]]
```

Figure 7: Τα αποτελέσματα της αναζήτησης

Το σύστημα συγκρίσεων

Ολοκληρώνοντας την εφαρμογή μας προσθέτουμε μια τελευταία συνάρτηση. Πιθανόν ο χρήστης να θέλει να μάθει πόσο όμοιες κρίνει το μοντέλο μας δυο ταινίες. Δημιουργούμε λοιπόν μια συνάρτηση η οποία δέχεται δυο ταινίες για είσοδο. Στην συνέχεια εκτυπώνει το ποσοστό ομοιότητας τους, ακολουθούμενο από τα πρώτα δέκα πιο συνηθισμένα tokens στο έγγραφο της κάθε ταινίας και το σημαντικότερο θέμα τους.

In [139]:

```
def movie_comparer(title1, title2):
    movies_checked = 0
    for i in range(len(movies_clean)):
        similarity = []
        if movies_clean.iloc[i][1] == title1:
            lda_vectors1 = corpus_lda_model[i]
            sims = index[lda_vectors1]
            for j in range(len(movies_clean)):
                if movies_clean.iloc[j][1] == title2:
                    lda_vectors2 = corpus_lda_model[j]
                    similarity.append(sims[j])
                    print('The similarity between {} and {} is: {}'.format(title1, title2, similarity[0]))

                    print('-----')
                    print("{}'s most prominent tokens are:".format(title1))
                    article_tokens = corpus[i]
                    sorted_tokens = sorted(article_tokens, key=lambda x: x[1], reverse=True)
                    sorted_tokens_10 = sorted_tokens[:10]
                    for i in range(len(sorted_tokens_10)):
                        print("Word {} (\"{}\") appears {} time(s).".format(sorted_tokens_10[i][0],
                                                                            dictionary[sorted_tokens_10[i][0]],
                                                                            sorted_tokens_10[i][1]))

                    print('-----')
                    print("{}'s most prominant topic is:")
                    print(lda_model_final.print_topic(max(lda_vectors1, key=lambda item: item[1])[0]))

                    print('-----')
                    print("{}'s most prominent tokens are:".format(title2))
                    article_tokens = corpus[j]
                    sorted_tokens = sorted(article_tokens, key=lambda x: x[1], reverse=True)
                    sorted_tokens_10 = sorted_tokens[:10]
                    for i in range(len(sorted_tokens_10)):
                        print("Word {} (\"{}\") appears {} time(s).".format(sorted_tokens_10[i][0],
                                                                            dictionary[sorted_tokens_10[i][0]],
                                                                            sorted_tokens_10[i][1]))

                    print('-----')
                    print("{}'s most prominant topic is:")
                    print(lda_model_final.print_topic(max(lda_vectors2, key=lambda item: item[1])[0]))
```

In [156]:

```
movie_comparer("Eye in the Sky", "Pompeii")
```

```
The similarity between Eye in the Sky and Pompeii is: 0.719353437423706
-----
Eye in the Sky's most prominent tokens are:
Word 1148 ("alejandro") appears 23 time(s).
Word 1964 ("kate") appears 23 time(s).
Word 332 ("s") appears 15 time(s).
Word 1078 ("team") appears 14 time(s).
Word 390 ("tell") appears 10 time(s).
Word 9599 ("diaz") appears 10 time(s).
Word 1341 ("gun") appears 9 time(s).
Word 6057 ("silvio") appears 9 time(s).
Word 2308 ("mexican") appears 8 time(s).
Word 6016 ("fausto") appears 8 time(s).
-----
{}'s most prominant topic is:
0.029*"jim" + 0.020*"s" + 0.012*"gang" + 0.010*"simon" + 0.008*"member" + 0.008*"find" + 0.008*"wayn" + 0.007*"mcclane" + 0.007*"get" +
0.007*"kate"
-----
Pompeii's most prominent tokens are:
Word 1717 ("jim") appears 90 time(s).
Word 1805 ("gang") appears 33 time(s).
Word 332 ("s") appears 30 time(s).
Word 247 ("member") appears 25 time(s).
Word 6301 ("wayn") appears 25 time(s).
Word 143 ("find") appears 18 time(s).
Word 1072 ("student") appears 17 time(s).
Word 162 ("get") appears 16 time(s).
Word 29 ("back") appears 15 time(s).
Word 1848 ("train") appears 15 time(s).
-----
{}'s most prominant topic is:
0.029*"jim" + 0.020*"s" + 0.012*"gang" + 0.010*"simon" + 0.008*"member" + 0.008*"find" + 0.008*"wayn" + 0.007*"mcclane" + 0.007*"get" +
0.007*"kate"
```

Figure 8: Τα αποτελέσματα της σύγκρισης

Τα συμπεράσματα

Ξεκινώντας την έρευνα για να βρούμε τον τρόπο υλοποίησης της εφαρμογής αυτής, στόχος ήταν επιπλέον να χρησιμοποιηθούν εξτρά δεδομένα όπως σκηνοθέτες, είδος, παραγωγοί, έτος κυκλοφορίας, καστ κ.α. Δυστυχώς η φύση του αλγορίθμου είναι τέτοια που βασίζεται στο πλήθος και την επανάληψη των λέξεων για να αποδώσει σωστά. Από τα αποτελέσματα που λαμβάνουμε συμπεραίνουμε ότι ο αλγόριθμος αποδίδει ικανοποιητικά στην κατηγορία των ταινιών. Λόγω του μικρού εύρους της σύνοψης κάθε ταινίας, δεν αποδίδει όσο καλά αποδίδει ο αλγόριθμος σε βιβλία και βιβλιογραφικές εργασίες.

Συνεπώς σε μια μεταγενέστερη μορφή της εφαρμογής προτείνουμε να χρησιμοποιηθούν τα σενάρια κάθε ταινίας αντί για την σύνοψη. Με αυτή την αλλαγή υποθέτουμε ότι τα αποτελέσματα θα είναι βελτιωμένα διότι το κείμενο θα πλησιάζει την μορφή ενός βιβλίου. Η εργασία βασίζεται στην βιβλιογραφική εργασία "Recommendation system based on semantic scholar mining and topic modeling on conference publications".