Data Mining Techniques

Εργασία 1

Παναγιώτης Καψάλης DS3516005
pkapsalis@aueb.gr

Τα αρχεία κώδικα που αποτελούν την υλοποίηση της εργασίας είναι τα εξής:

code.py: περιέχει την συνάρτηση exercise() που δέχεται σαν ορίσματα το id του κειμένου για το οποίο θέλουμε να βρούμε με Jaccard ομοιότητες, το πλήθος των Hash functions που πρόκειται να χρησιμοποιηθούν, οι οποίοι ειναι οι n γείτονες του κειμένου που πρόκειται να επιστραφούν.

prime_numbers.py: περιέχει την συνάρτηση prime(), η οποία χρησιμοποιείται για τον υπολογισμό του αριθμού c, ο οποίος είναι ο αμέσως επόμενος αριθμός από τον συνολικό αριθμό των shingles.

random_numbers.py: περιέχει την συνάρτηση randomNums(), η οποία χρησιμοποιείται για τον υπολογισμό τυχαίων διαφορετικών ακεραίων αριθμών a και b, οι οποίοι θα χρησιμοποιηθούν από την οικογένεια hash functions **h(x) = (ax+b)modc** .

Στα πλαίσια της πρώτης εργασίας του μαθήματος Data Mining, πρέπει να υπολογίσουμε ομοιότητα μεταξύ κειμένων. Τα κείμενα τα κατεβάσαμε από το UCI Machine Learning Repository. Είναι 22 κείμενα σε μορφή .sgm, κάθενα από αυτά περιέχει html documents, τα οποία τα διακρίνουμε από τα tags BODY και REUTERS πότε αρχίζει και τελειώνει το καθένα. Πιο συγκεκριμένα με την χρήση της βιβλιοθήκης beautiful soup της Python, εξάγουμε κείμενο που βρίσκεται μεταξύ των tags "REUTERS", τα κείμενα τα αποθηκεύουμε σε μια λίστα. Το παρακάτω κομμάτι κώδικα προέρχεται από το αρχείο code.py, υλοποιεί την συνάρτηση exercise(), και σε κάθε .sgm αρχείο απογυμνώνει το κείμενο από τα tags “REUTERS” και εκτυπώνει στο τέλος το συνολικό πλήθος των html κειμένων. Στο παρακάτω κομμάτι κώδικα χρησιμοποιείται το directory, που έχω τα .sgm files.

```
from bs4 import BeautifulSoup,SoupStrainer
import random
import re
from prime_numbers import prime
from random_numbers import randomNums
def exercise(k, num, n,doc_id):
    ##Get HTML Body from .sgm files
    text = []
    for i in range(22):
        if i < 10:
            print('reut2-00'+str(i)+'.sgm')
            f = open('C:/Users/User/Desktop/Data Mining/reut2-00'+str(i)+'.sgm', 'r')
            data= f.read()
            soup = BeautifulSoup(data)
            contents = soup.findAll('reuters')
            for reuter in contents:
                text.append(reuter.text)

        else:
            print('reut2-0'+str(i)+'.sgm')
            f = open('C:/Users/User/Desktop/Data Mining/reut2-0'+str(i)+'.sgm', 'r')
            data= f.read()
            soup = BeautifulSoup(data)
            contents = soup.findAll('reuters')
            for reuter in contents:
                text.append(reuter.text)
    print('Ta sunolika documents einai:', len(text))
```

Τα συνολικά κείμενα λοιπόν τα οποία εξάγουμε με την χρήση της βιβλιοθήκης beautiful soup, από τα αρχεία είναι **21578** κείμενα. Στο επόμενο βήμα απογυμνώνουμε τα κείμενα από special characters (τελείες, θαυμαστικά, html parts), με χρήση regural expressions, και στην συνέχεια για να σπάσουμε το κάθε κείμενο στις λέξεις του, χρησιμοποιούμε την συνάρτηση split. Ο κώδικας που χρησιμοποιήθηκε για αυτό είναι ο παρακάτω:

```
print('Ta sunolika documents einai:', len(text))
##from documents remove special characters and split each documents to words
text = [re.sub(r'[^\w]', ' ', elem).lower() for elem in text]
splited_text = [doc.split() for doc in text]
```

Αφού ολοκληρωθεί, η διαδικασία μετατροπής των κειμένων σε λέξεις, ερχόμαστε στο στάδιο εφαρμογής της τεχνικής k-singles. Σύμφωνα με την εκφώνηση της εργασίας, αφού ο χρήστης εισάγει τον αριθμό k του πλήθους των singles, χωρίζουμε τις λέξεις του κάθε κειμένου σε σύνολα k πλήθους λέξεων που εμφανίζονται στο κείμενο. Στην συνέχεια μετράμε το πλήθος όλων των singles. Ουσιαστικά ορίζουμε τα K-singles σαν μια σειρά k λέξεων του κειμένου κάθε φορά. Και αποθηκεύουμε τα δεδομένα με τέτοιο τρόπο έτσι ώστε, να έχουμε μια list of lists, όπου κάθε λίστα περιέχει τα singles του κάθε κειμένου. Ο παρακάτω κώδικας υλοποιεί αυτά που περιγράψαμε.

```
##find shingles per text....we want to split each text's words
#k = int(input("Enter k: "))
s_test = []
shingles = []
for t in splited_text:
    temp = []
    for i in range(len(t)):
        if i != len(t):
            shingles.append(''.join(t[i:k+i]))
            temp.append(''.join(t[i:k+i]))
    s_test.append(temp)
number_unique_shingles = len(set(shingles))
print('To plh8os twn monadikwn k-shingles einai:', number_unique_shingles)
total_shingles = len(shingles)
print('To sunoliko plh8os twn k-shingles einai:', total_shingles)
unique_shingles = set(shingles)
```

Έπειτα, δημιουργούμε ένα dictionary, έτσι ώστε να δώσουμε ένα id στα μοναδικά singles. Πιο συγκεκριμένα η τεχνική των k-singles βασίζεται στην λογική ότι κείμενα θεωρούνται όμοια όταν έχουν πολλά singles, k-αλληλουχίες λέξεων για εμάς, ίδια ακόμα και αν βρίσκονται σε διαφορετική σειρά. Επίσης η τεχνική αυτή θεωρείται και μια τεχνικής συμπίεσης των αρχικών κειμένων.

```
##create dictionary, to save unique shingles and create id for these shingles
shingles_dict = dict()
shingles_list = list(unique_shingles)
j = 0
for item in shingles_list:
    j+=1
    shingles_dict[item] = j
print('Dhmiourgia dictionary poy exei ws kleidia ta monadika shingles kai ws times ta antistoixa id tous')
#print(shingles_dict)
```

Στο επόμενο βήμα βρίσκουμε ποια shingles εμφανίζονται σε κάθε κείμενο, δημιουργούμε την λιστα s_test_id, η οποία είναι μια List of lists, δηλαδή για κάθε κείμενο έχει μια λίστα με τα id των shingles που εμφανίζονται σε αυτό. Για παράδειγμα το στειχείο s_test_id[0] είναι μια λίστα που περιέχει τα id των shingles που εμφανίζονται στο πρώτο κείμενο, το στοιχείο s_test_id[1] είναι μια λιστα που περιέχει τα id των shingles που εμφανίζονται στο δεύτερο κείμενο.

```
#print(shingles_dict)
## With the following code, we will append the shingles id's to documents that have the specific shingles
s_test_id = []
for text in s_test:
    temp1 = []
    for sh in text:
        if sh in shingles_dict.keys():
            temp1.append(shingles_dict[sh])
    s_test_id.append(temp1)
print('Dhmiourgh8hke h lista me tis listes twn ids twn shingles poy periexei ka8e keimeno')
#print(s_test_id)
```

Σε αυτό το σημείο πρέπει να αναφέρουμε το εξής, ότι η σύγκριση των κειμένων θα βασιστεί στην Jaccard Similarity, πιο συγκεκριμένα θα υπολογίζουμε το πλήθος της τομής των k-singles 2 κειμένων διαιρεμένο με το πλήθος της ένωσης των K-singles.

$$\textbf{Sim(doc1,doc2)} = |\textbf{doc1} \cap \textbf{doc2}|/|\textbf{doc1} \cup \textbf{doc2}|$$

Η σύγκριση όμως των k-singles των κειμένων, μπορεί να πάρει πάρα πολύ για αυτό, και θα υλοποιήσουμε MinHashing, θα κάνουμε hash τα k-singles των κειμένων έτσι ώστε να συκρίνουμε τις υπογραφές των κειμένων, καθώς σύμφωνα με την θεωρία ισχύει ότι η ομοιότητα 2 κειμένων doc1, doc2 ισούται με την ομοιότητα των υπογραφών των 2 κειμένων, δηλαδή:

$$\textbf{Sim(doc1,doc2)=Sim(Sig(doc1),Sig(doc2))}$$

Σύμφωνα με τα δεδομένα της άσκησης θα παράξουμε hash functions Οι οποίες θα ανήκουν στην οικογένεια h(x) = (ax+b)modc, όπου c είναι ο μεγαλύτερος πρώτος αριθμός από τον συνολικό αριθμό των singles. Η συνάρτηση prime(), βρίσκεται στο αρχείο prime_number.py, υπολογίζει τον αμέσως επόμενο πρώτο αριθμό μετά το συνολικό πλήθος των singles. Με τον παρακάτω κώδικα καλούμε την συνάρτηση prime() για τον υπολογισμό του αριθμού c.

```
##maximum shingle id, that was assigned
maxShingle_id = max(shingles_dict.values())
## find nearest prime number (c)
c = maxShingle_id
response = False
while(response == False):
    c = c + 1
    response = prime(c)
print('The number c for hash function family is :',c)
```

Στην συνέχεια ο χρήστης δίνει ως όρισμα το πλήθος των hash συναρτήσεων, που θα χρησιμοποιηθούν. Η συνάρτηση randomNums, κάνοντας χρήση την βιβλιοθήκη random της Python, δημιουργεί κ,όσο το πλήθος των hash functions, ακεραίους άριθμούς διαφορετικούς ο ένας από τον άλλον. Έτσι λοιπόν δημιουργούμε τα a,b οι οποίοι είναι οι σταθερές των k διαφορετικών hash functions.

```
## generate k hash functions
#num = int(input("Set number of hash functions:  "))
a = randomNums(num, maxShingle_id)
b = randomNums(num, maxShingle_id)
```

Εφόσον έχουμε συλλέξει όλα τα id των μοναδικών shingles, και τα έχουμε αποθηκεύσει σε ένα dictionary, θα εφαρμόσουμε τις hash functions πάνω στα shinles id's των κειμένων και στην συνέχεια θα πάρουμε το ελάχιστο hash code value. Έτσι με αυτή την τεχνική εξάγουμε τις υπογραφές, signatures, των κειμένων. Ουσιαστικά οι υπογραφές των κειμένων είναι Οi hash τιμές των id's των singles.

```
    ##test for one document, let it be the first document
    signatures = []
    for doc in s_test_id:
        signature = []
        #for each of the hash functions
        for i in range(0, num):
            #for each of the shingles, calculate it's hash code using i-th hash function
            minHashCode = c + 1
            for shId in doc:
                hashCode = (a[i]*shId + b[i])%c
                # Track the lowest hash code seen.
                if hashCode < minHashCode:
                    minHashCode = hashCode
            signature.append(minHashCode)
        signatures.append(signature)
    print('Oi upografes twn keimenwn dhmiourgh8hkan')
    #print(signatures)
```

Στο επόμενο βήμα υπολογίζουμε τις Jaccard Similarities, μεταξύ των κειμένων. Ο χρήστης δίνει σαν είσοδο το id του κειμένου για το οποίο θέλουμε να βρούμε όμοια κείμενα. Υπολογίζουμε το Jaccard Similarity, σύμφωνα με τους παραπάνω τύπους και επιστρέφουμε τα n, πλήθος των n πιο κοντινών κειμένων στο κείμενο του οποίου το id έδωσε σαν όρισμα ο χρήστης. Το πρόγραμμα επιστρέφει ένα λεξικό, στο οποίο έχει ως κλειδιά τα n πιο κοντινά κείμενα και σαν dictionary values τις Jaccard Similarities των κειμένων με το δοσμένο από τον χρήστη κείμενο.

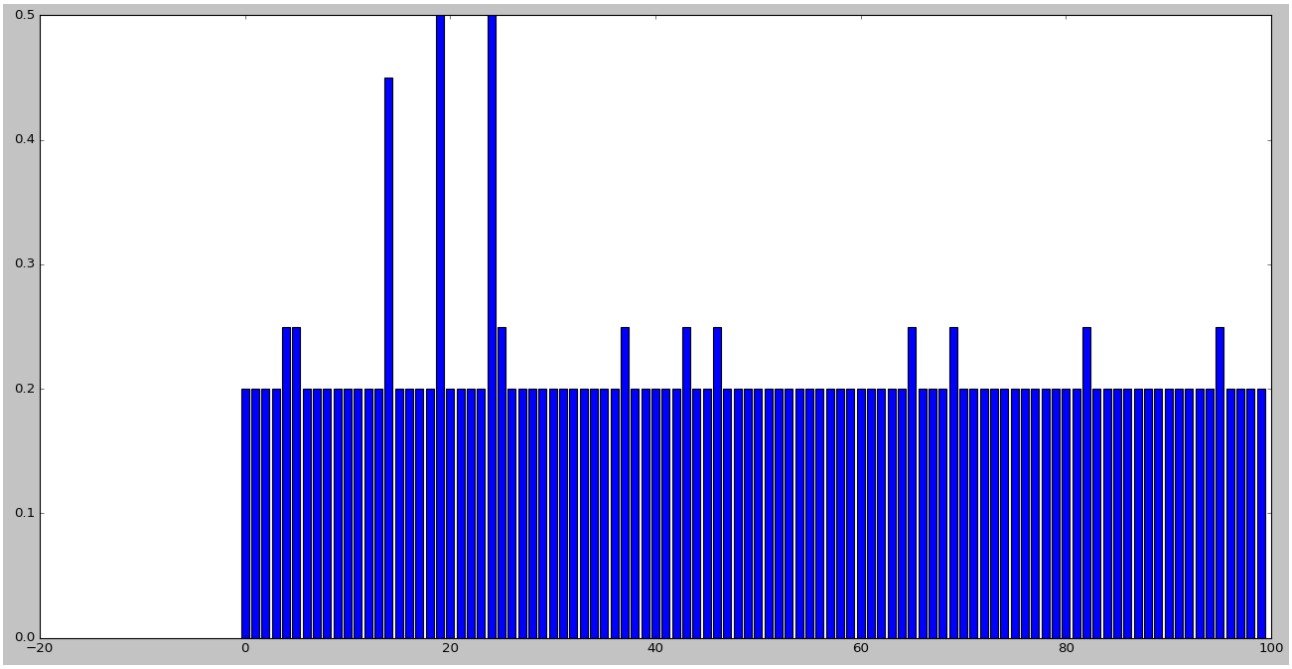
```
    #print(signatures)
    ##Next step....Find Jaccard Similarities
    ## we return a dictionary, and as key we have the document compared with the selected document, and as value we have
    ##the jaccard similarity of selected document with the other documents
    ## Finaly we return the top n maximum similarities
    #doc_id = int(input("Set document id to check jacard similarities:  "))
    #n = int(input("Set number of closest documents to return:  "))
    doc_sign = signatures[doc_id]
    similarities = dict()
    for l in range(len(signatures)):
        if l == doc_id:
            continue
        else:
            jac_sim = len(set(doc_sign)&set(signatures[l]))/len(doc_sign + signatures[l])
            similarities[str(l)] = jac_sim
    t = sorted(similarities.items(), key=lambda x:-x[1])[:n]
    return t
```

Η συνάρτηση στο τέλος επιστρέφει μια λίστα tuples όπου σε κάθε tuple είναι οι κοντινότεροι γείτονες και η Jaccard Similarity του συγκεκριμένου γείτονα με το δοσμένο κείμενο, για παράδειγμα η παρακάτω εικόνα αποτελεί μια κλήση της συνάρτησης exercise()

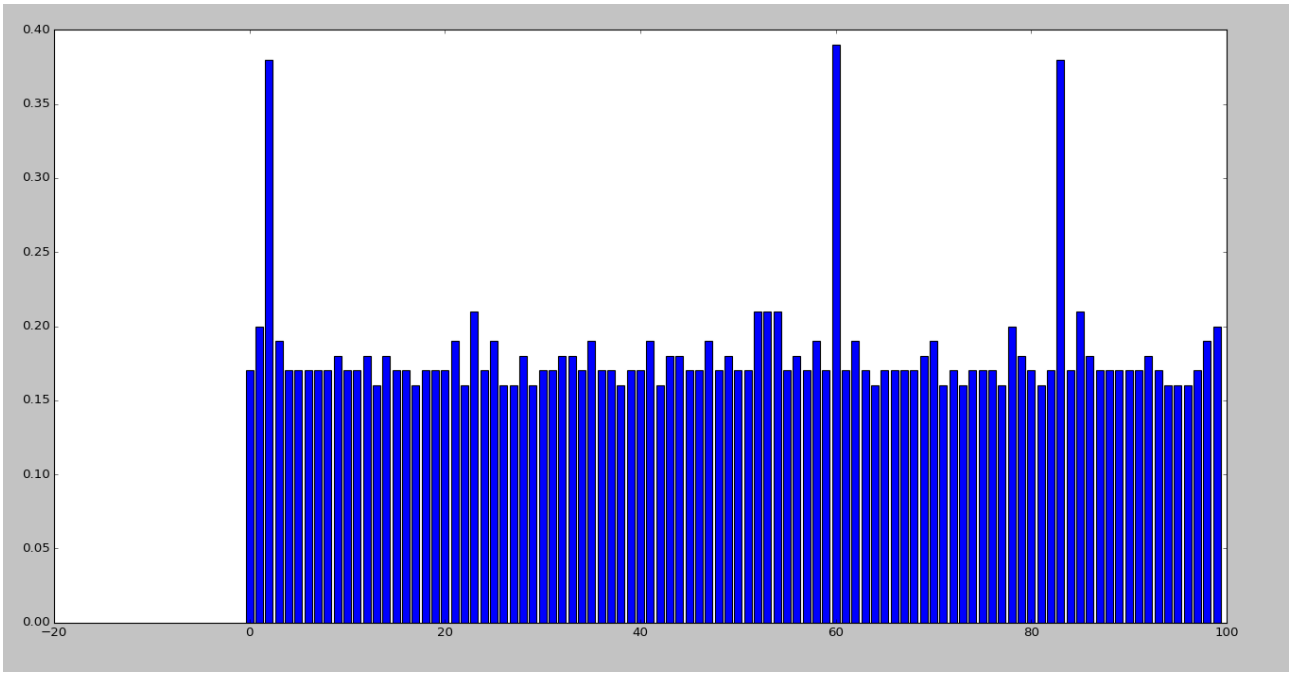
```
Out[11]: [('2664', 0.08),
          ('10495', 0.07),
          ('1414', 0.07),
          ('2368', 0.07),
          ('13206', 0.07),
          ('6799', 0.07),
          ('10151', 0.07),
          ('1817', 0.06),
          ('8380', 0.06),
          ('16042', 0.06),
          ('17799', 0.06),
          ('8537', 0.06),
          ('10841', 0.05),
          ('8903', 0.05),
          ('19345', 0.05),
          ('15333', 0.05),
```

Πιο συγκεκριμένα το πρώτο tuple της λίστας σημαίνει ότι το κείμενο με id 2664 έχει Jacaard Similarity με το κειμενο για το οποίο έχει δώσει id ο χρήστης.
Το αρχείο run.py καλεί την συνάρτηση exercise() και δίνει την δυνατότητα στον χρήστη να δόσει ορίσματα στην συνάρτηση exercise().

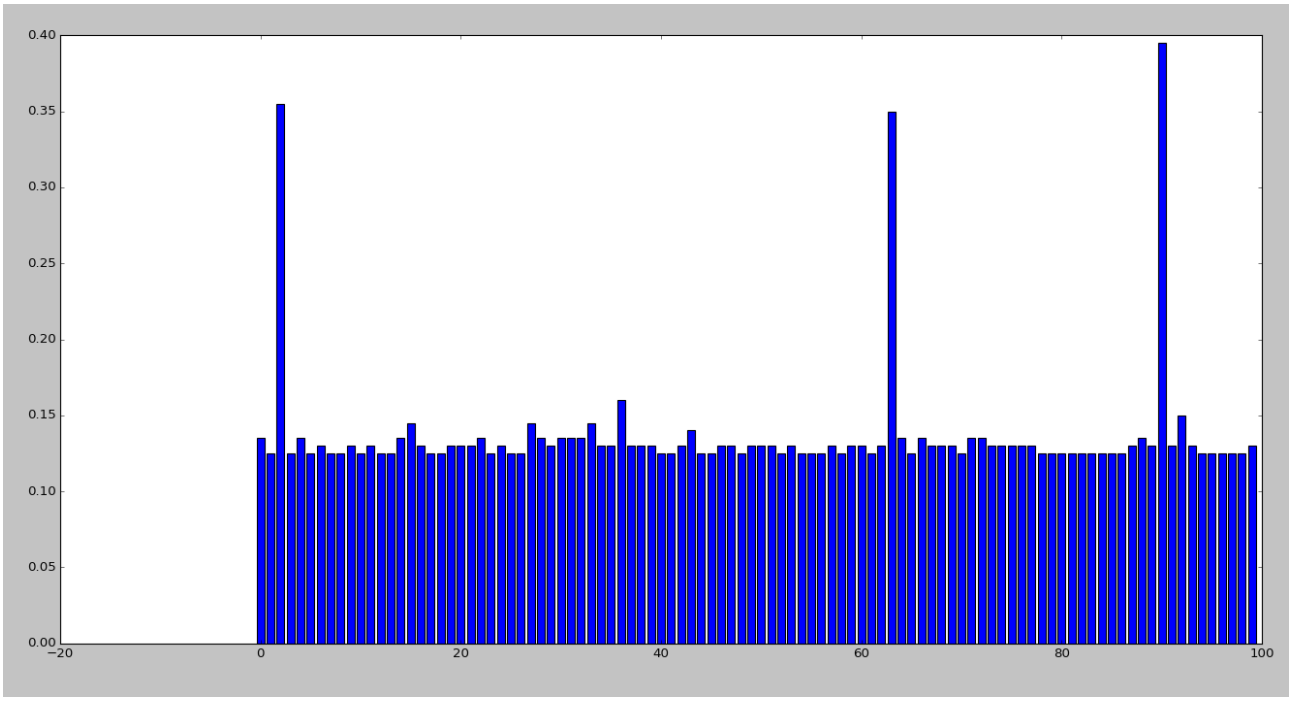
Στην συνέχεια εξατάζουμε την ακρίβεια των υπολογισμών το πρόγραμμα run_plots.py καλεί την συνάρτηση exercise για k = 1,2,3,4,5 και για κάθε k δίνουμε ως όρισμα κάθε φορά πλήθος 10, 50 , 100, 150 hash functions. Δηλαδή πιο συγκεκριμένα κάθε φορά κρατάμε σταθερό k και αλλάζουμε το πλήθος των hash functions. Έτσι λοιπόν με την εκτέλεση του run_plots.py script εξετάζουμε την Jaccard Similarity του κειμένου 4 με τα 100 πιο γειτονικά σε αυτό κείμενα για για k = 1,2,3,4,5 και για κάθε k δίνουμε ως όρισμα κάθε φορά πλήθος 10, 50 ,100, 150 hash functions.



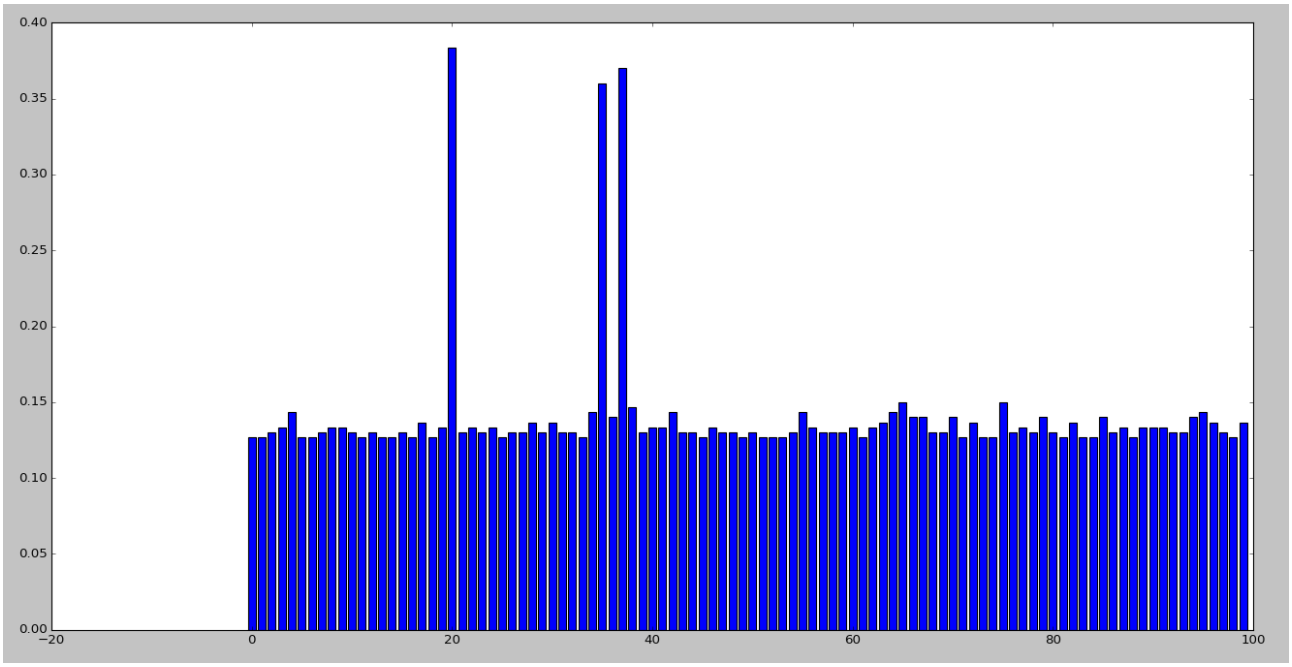
Εικόνα(1) k=1, 10 hash functions



Εικόνα(2) k=1, 50 hash functions

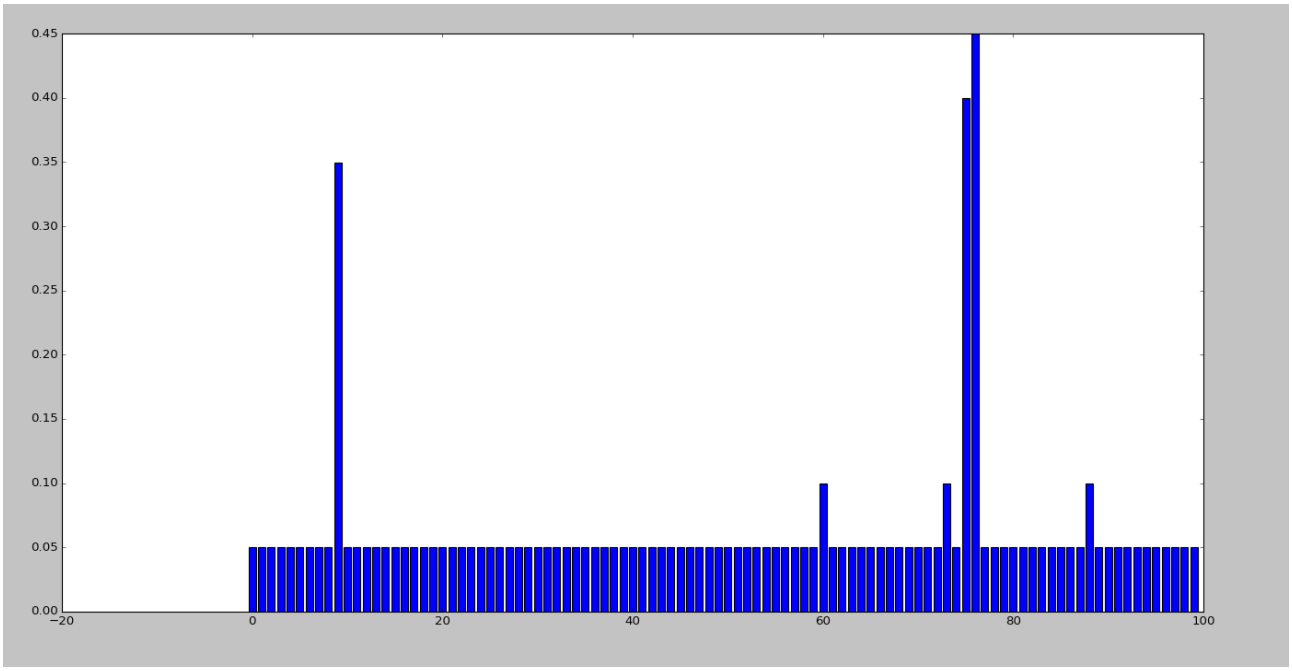


Εικόνα(3) k=1, 100 hash functions

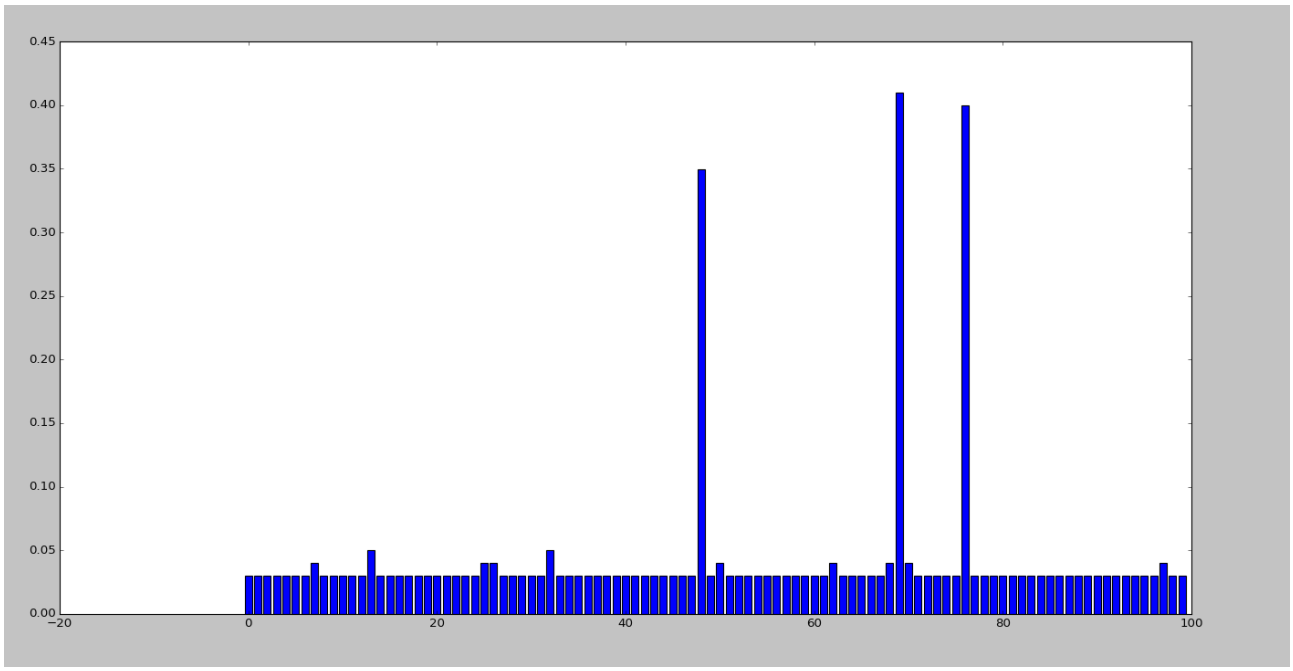


Εικόνα(4) k=1, 150 hash functions

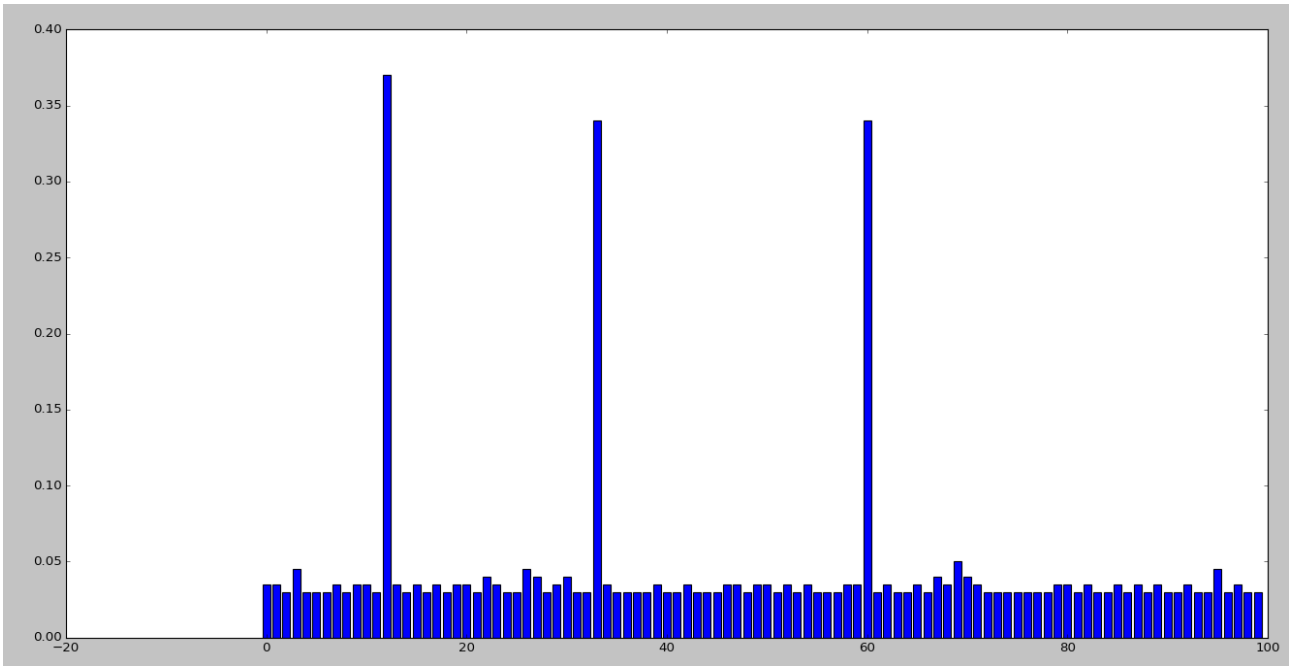
Παρατηρούμε ότι για k=1, και μεταβάλλοντας τον αριθμό των hash functions, ότι για μικρό αριθμό οι πιο κοντινοί γείτονες τείνουν να έχουν την ίδια ομοιότητα πράγμα λογικό καθώς όσο αυξάνουμε τον αριθμό των hash functions, υπάρχει δυνατότητα λιγότερα shingles να ανήκουν στην ίδια hash value. Γενικά παρατηρείται και οτί τα μικρότερα ποσοστά για k=1 είναι γύρω στα 13%



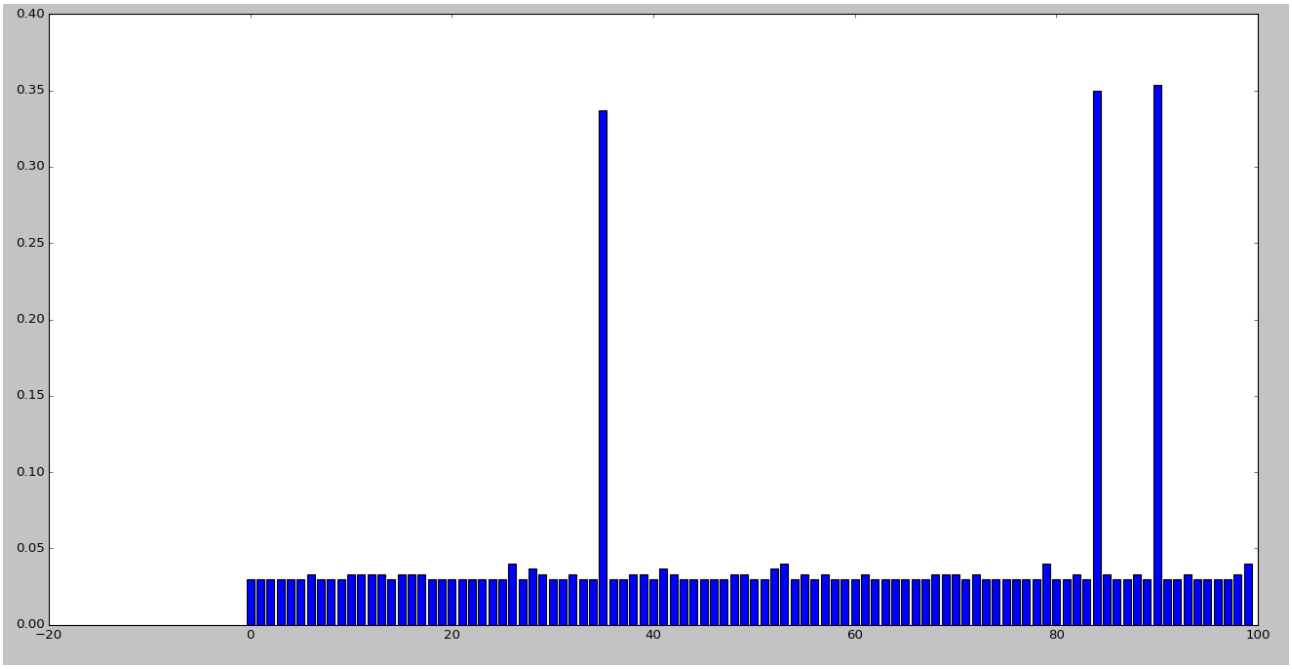
Εικόνα(5) k=2, 10 hash functions



Εικόνα(6) k=2, 50 hash functions

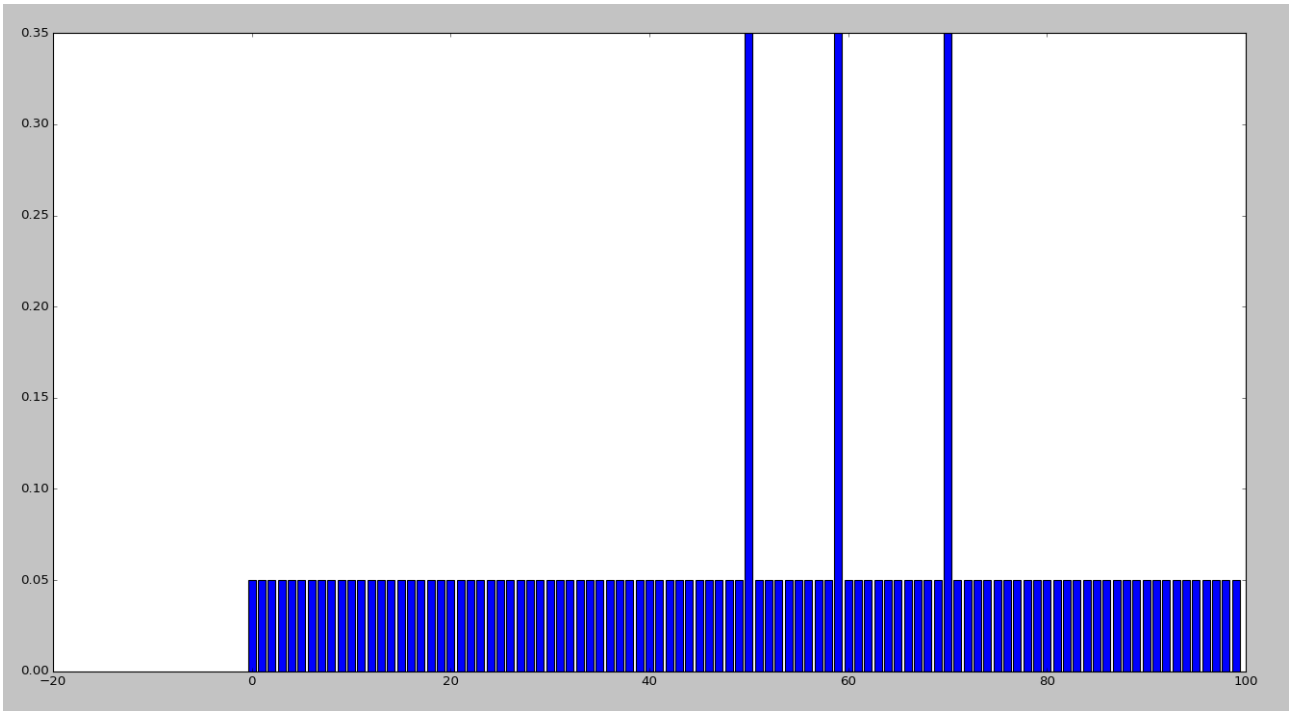


Εικόνα(7) k=2, 100 hash functions

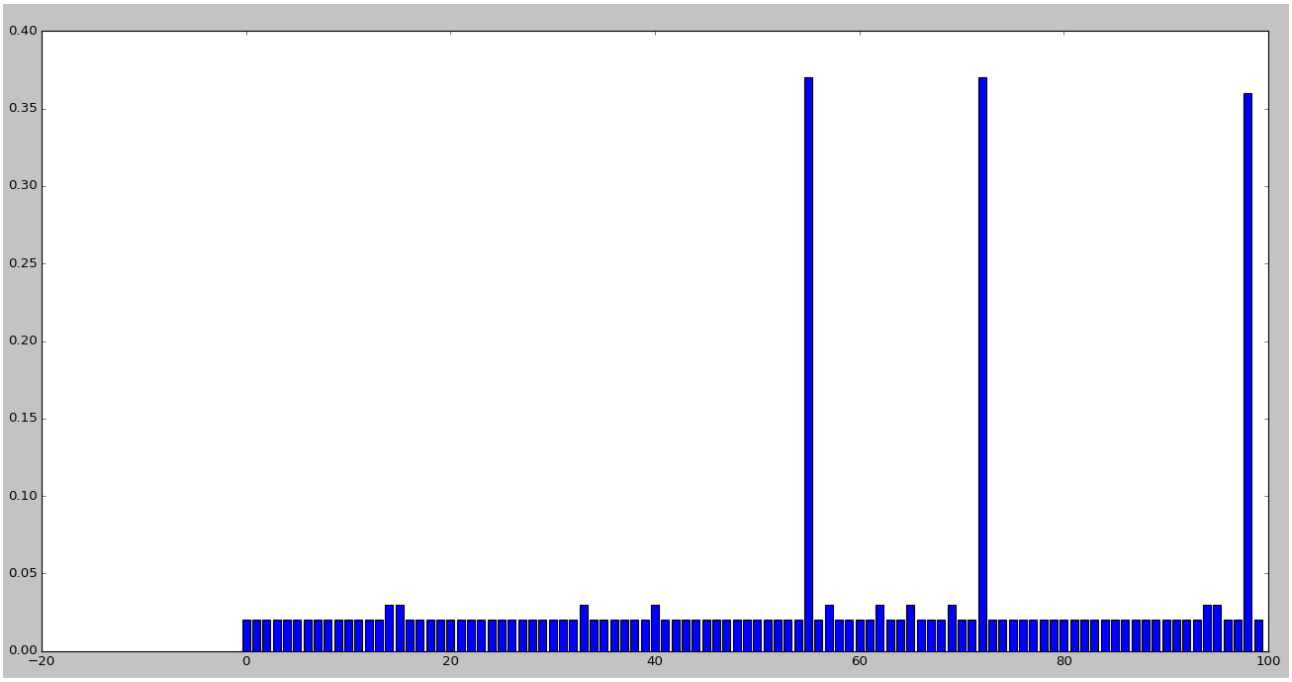


Εικόνα(8) k=2, 150 hash functions

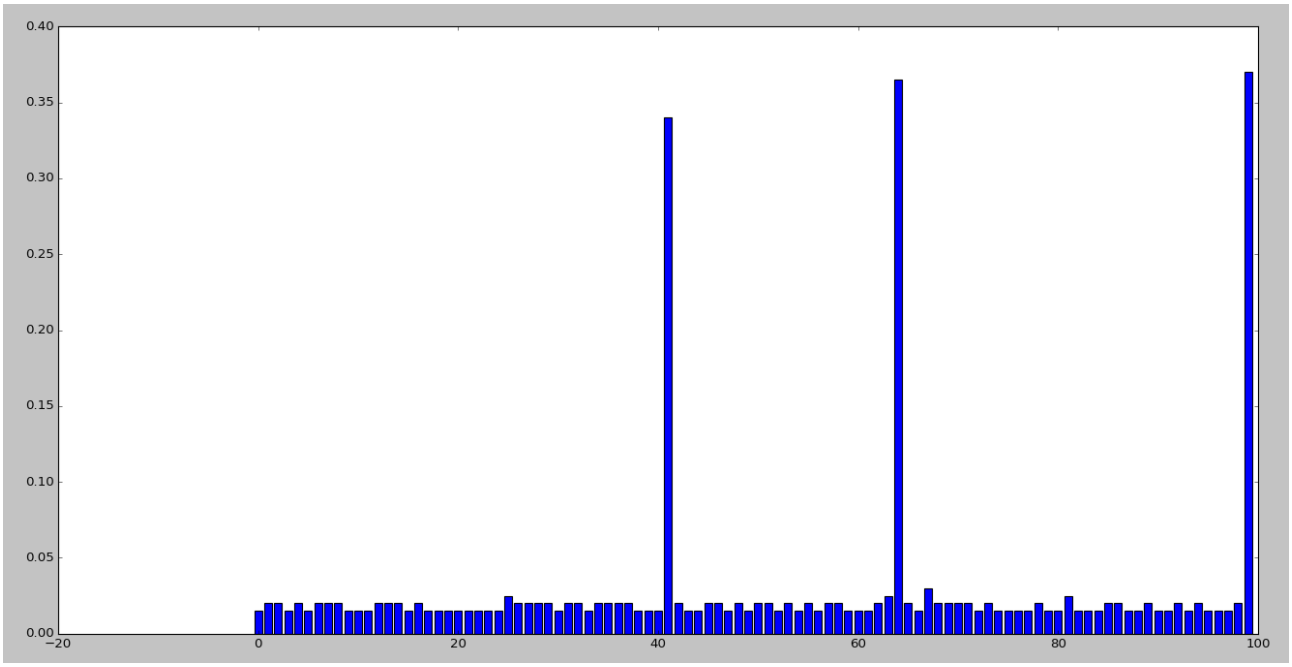
Για k=2 παρατηρούμε ότι η Jaccard ομοιότητα πέφτει σε τιμή, και είναι γύρω στα 5% εκτός από κάποια άλλα κείμενα που έχουν μεγαλύτερη ομοιότητα, παρατηρούμε πάλι ότι αυξάνοντας τον αριθμό των hash functions παρατηρούμε ότι έχουμε διαφορετικές τιμές της Jaccard ομοιότητας του κειμένου 4 με τα υπόλοιπα πιο γειτονικά κείμενα. Δηλαδή ότι όσο αυξάνουμε το πλήθος των συναρτήσεων τόσο η ομοιότητα πέφτει αριθμητικά. Επιπλέον ότι κάνοντας το κ =2, αυτόματα περιορίζεται πάλι η ομοιότητα καθώς να βρεθεί ενα 2-single σε 2 κείμενα είναι λιγότερο πιθανό από ότι για κ=1.



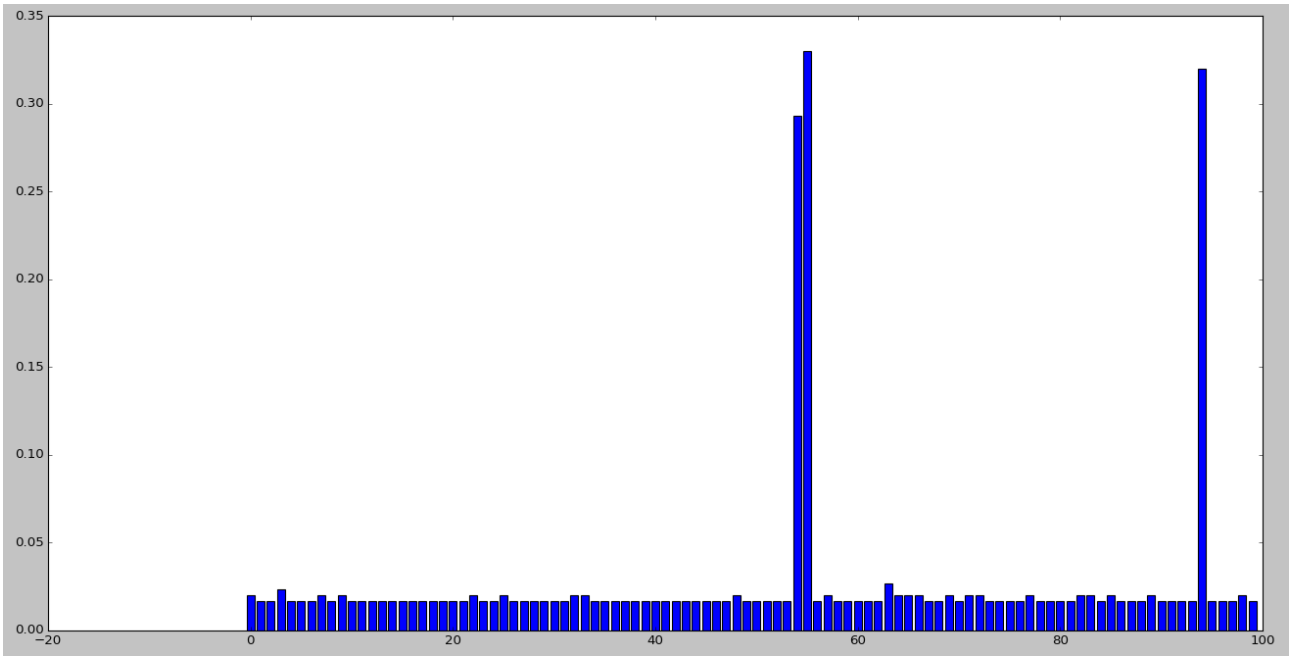
Εικόνα(9) k=3, 10 hash functions



Εικόνα(10) k=3, 50 hash functions

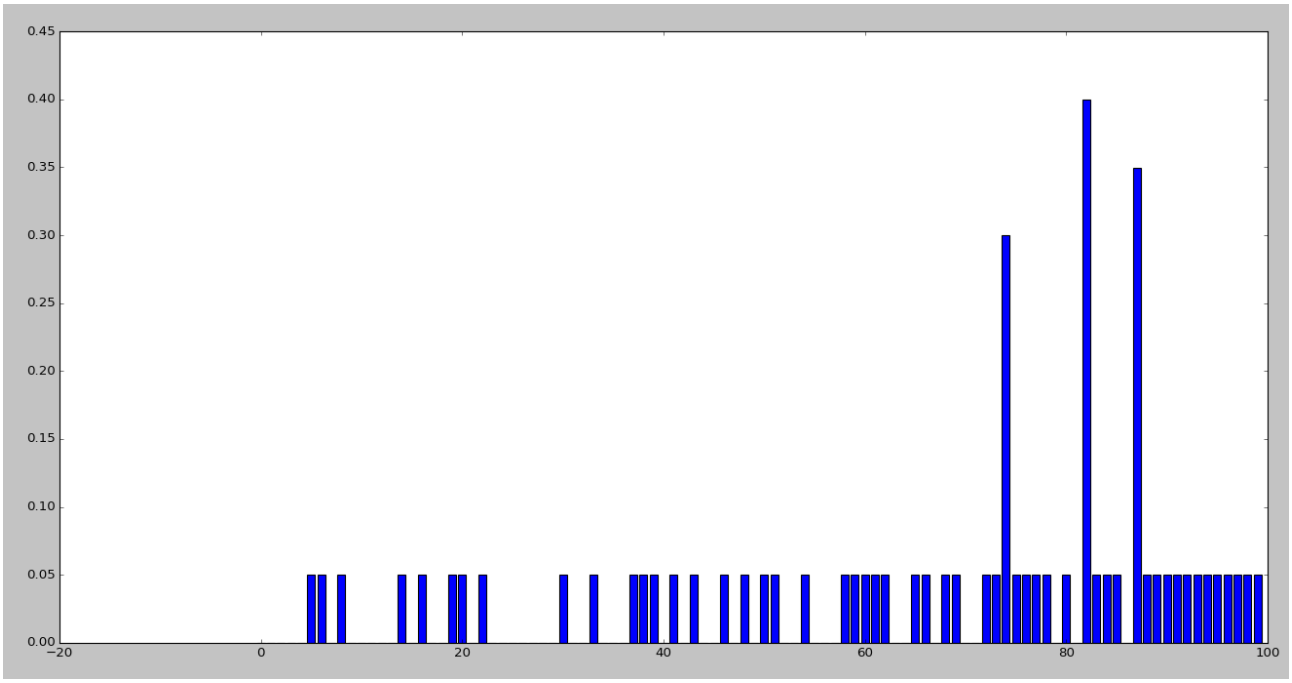


Εικόνα(11) k=3, 100 hash functions

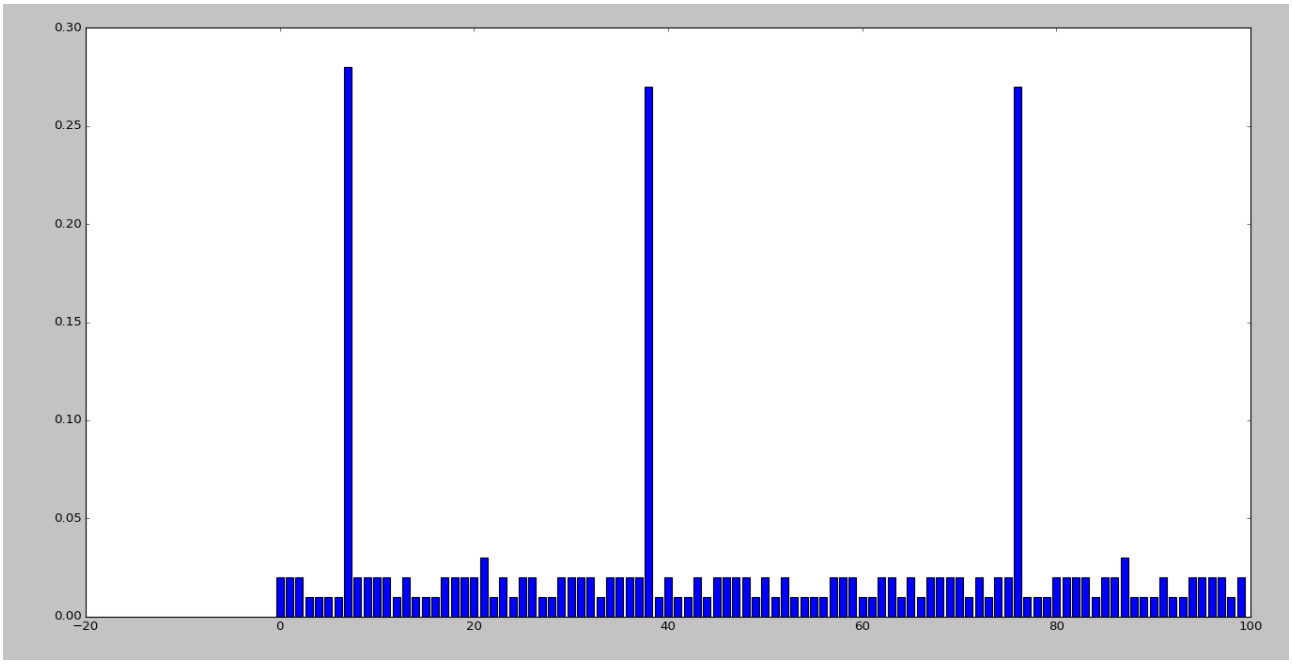


Εικόνα(12) k=3, 150 hash functions

Ομοίως για k=3 παρατηρούμε ότι η Jaccard ομοιότητα πέφτει σε τιμή, και είναι γύρω στα 2% εκτός από κάποια άλλα κείμενα που έχουν μεγαλύτερη ομοιότητα, παρατηρούμε πάλι ότι αυξάνοντας τον αριθμό των hash functions παρατηρούμε ότι έχουμε διαφορετικές τιμές της Jaccard ομοιότητας του κειμένου 4 με τα υπόλοιπα πιο γειτονικά κείμενα. Δηλαδή ότι όσο αυξάνουμε το πλήθος των συναρτήσεων τόσο η ομοιότητα πέφτει αριθμητικά. Επιπλέον ότι κάνοντας το κ =3, αυτόματα περιορίζεται πάλι η ομοιότητα καθώς να βρεθεί ενα 3-single σε 2 κείμενα είναι λιγότερο πιθανό από ότι για κ=1 ή για κ =2.

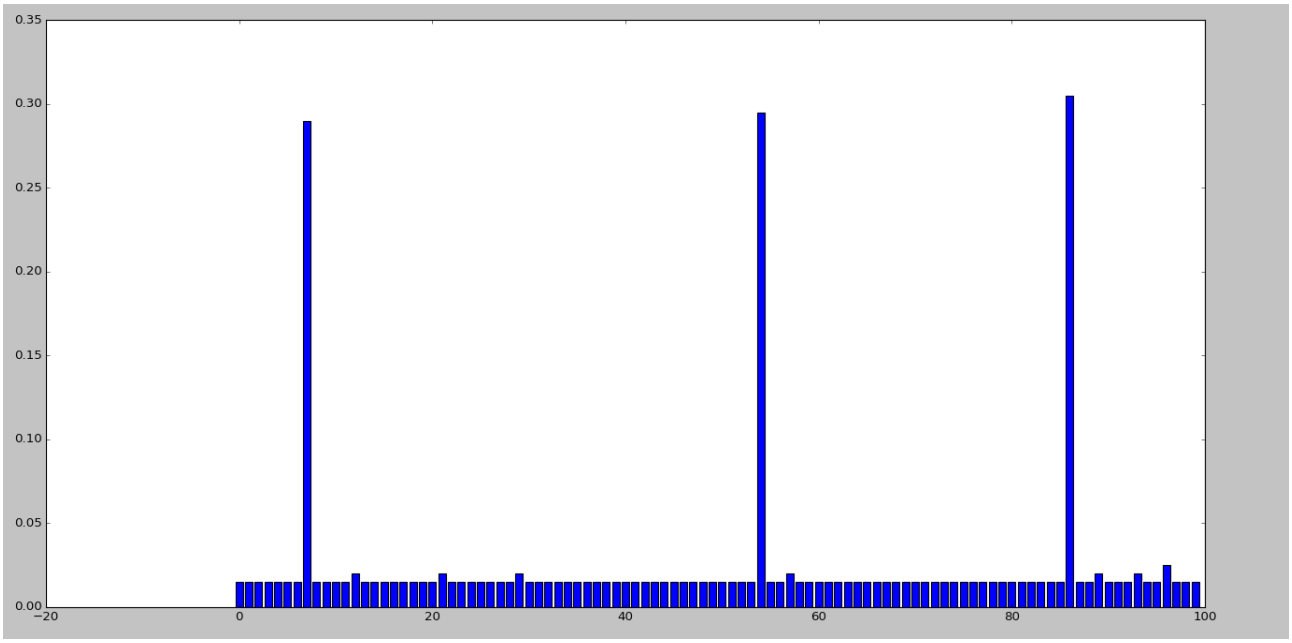


Εικόνα(13) k=4, 10 hash functions

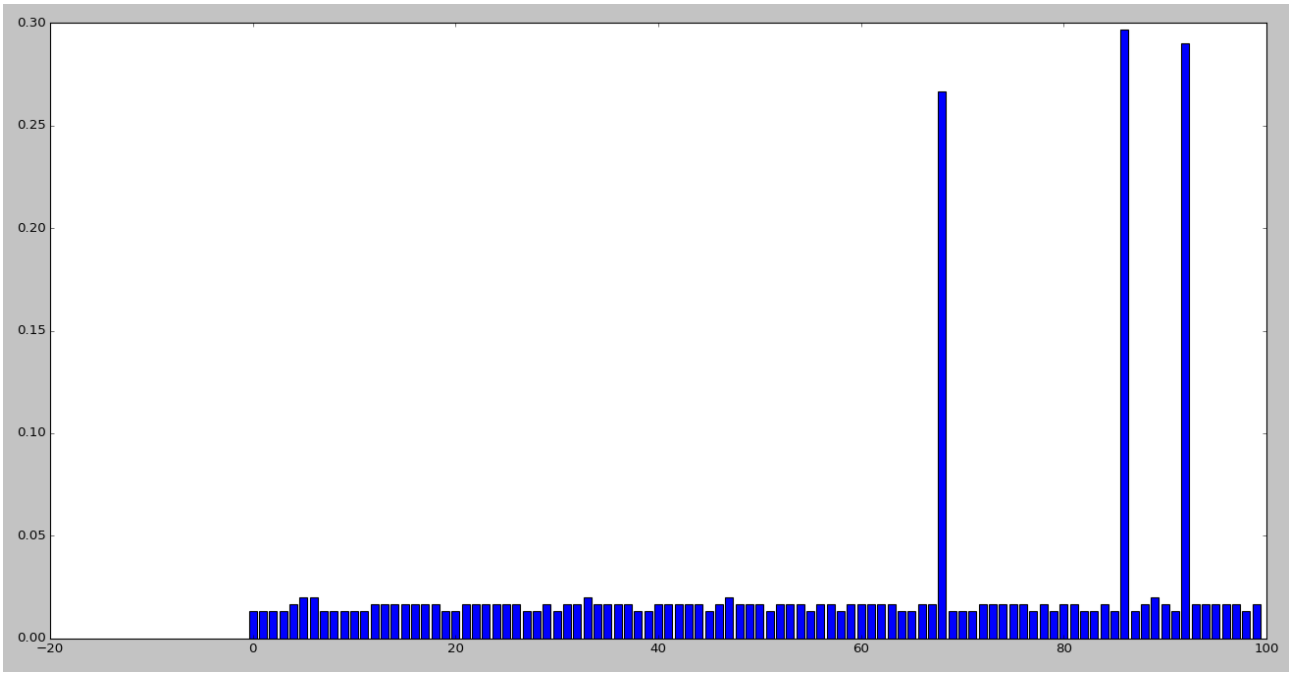


Εικόνα(14) k=4, 50 hash functions

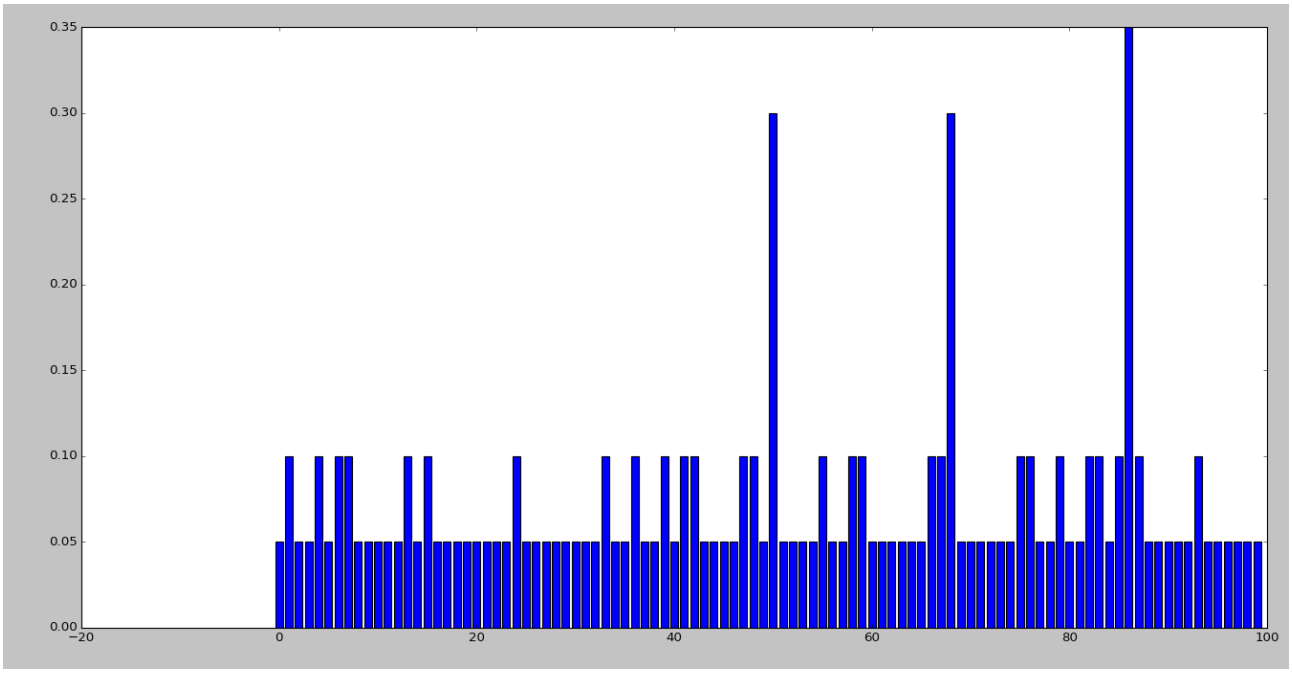
Για k=4 παρατηρούμε ότι η Jaccard ομοιότητα πέφτει σε τιμή, και είναι γύρω στα 2-3% εκτός από κάποια άλλα κείμενα που έχουν μεγαλύτερη ομοιότητα, παρατηρούμε πάλι ότι αυξάνοντας τον αριθμό των hash functions παρατηρούμε ότι έχουμε διαφορετικές τιμές της Jaccard ομοιότητας του κειμένου 4 με τα υπόλοιπα πιο γειτονικά κείμενα. Δηλαδή ότι όσο αυξάνουμε το πλήθος των συναρτήσεων τόσο η ομοιότητα πέφτει αριθμητικά. Επιπλέον ότι κάνοντας το κ =4, αυτόματα περιορίζεται πάλι η ομοιότητα καθώς να βρεθεί ενα 4-single σε 2 κείμενα είναι λιγότερο πιθανό από ότι για κ=1,2,3.



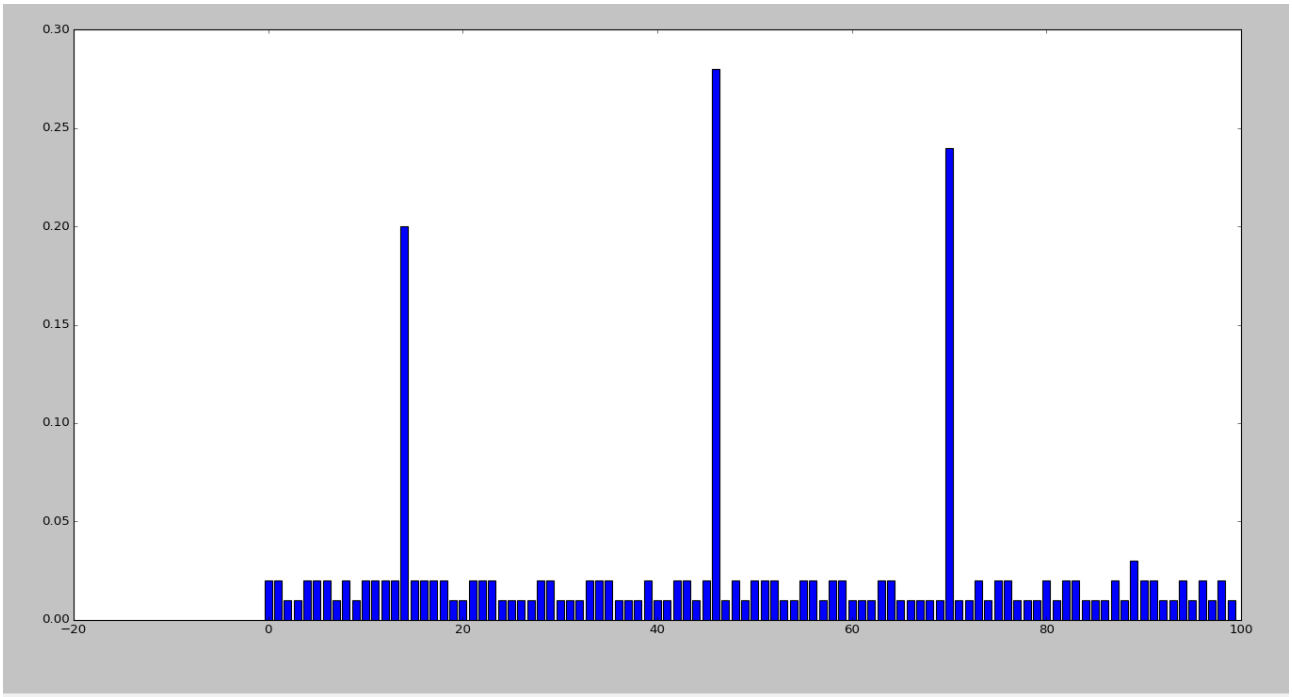
Εικόνα(15) k=4, 100 hash functions



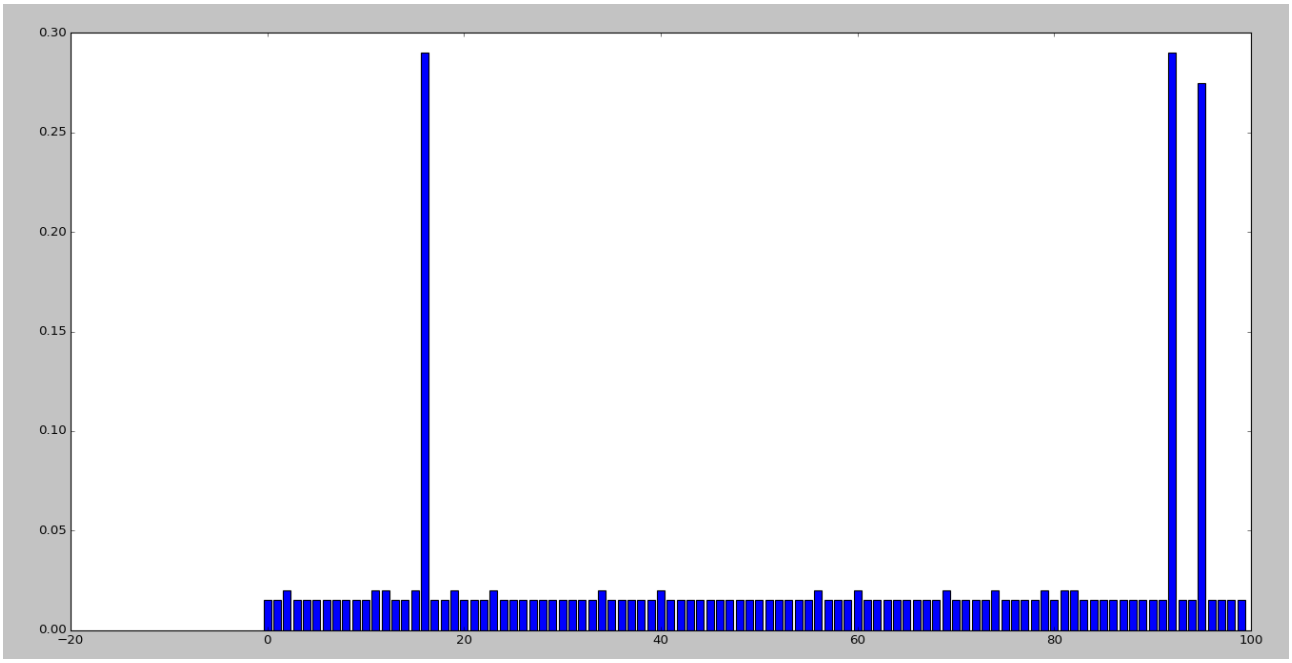
Εικόνα(16) k=4, 150 hash functions



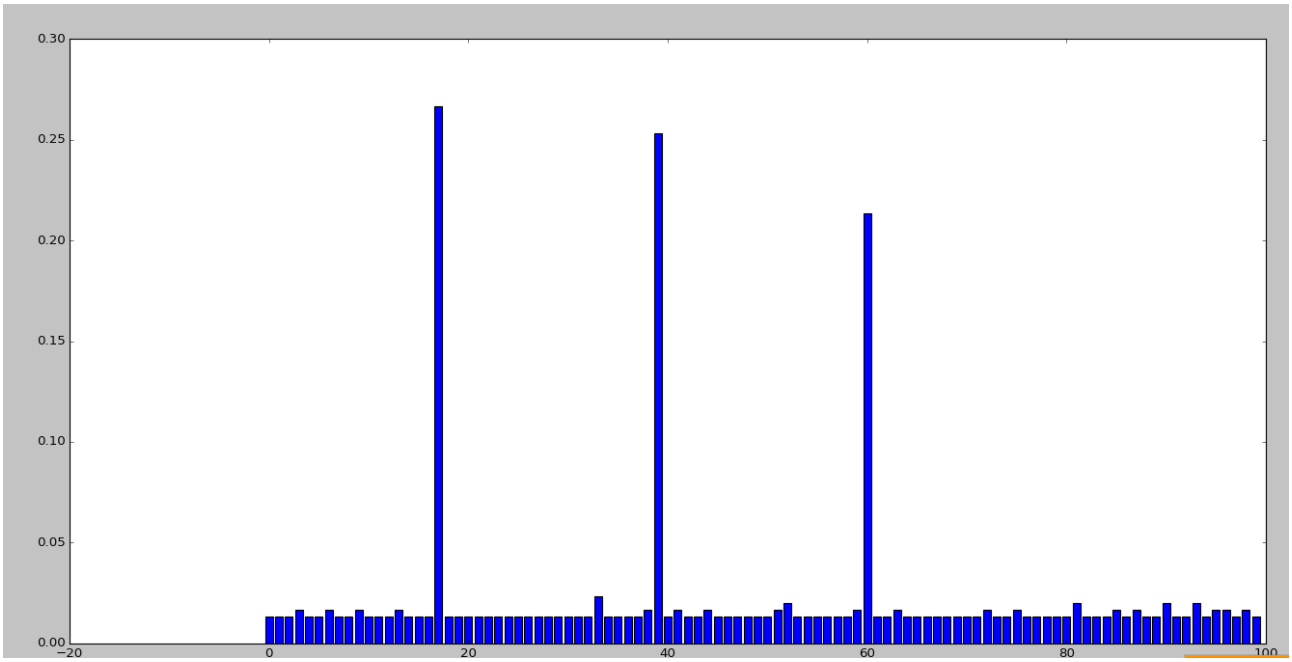
Εικόνα(17) k=5, 10 hash functions



Εικόνα(18) k=5, 50 hash functions



Εικόνα(19) k=5, 100 hash functions



Εικόνα(20) k=5, 150 hash functions

Για k=5 παρατηρούμε ότι η Jaccard ομοιότητα πέφτει σε τιμή, και είναι γύρω στα 1-2% εκτός από κάποια άλλα κείμενα που έχουν μεγαλύτερη ομοιότητα, παρατηρούμε πάλι ότι αυξάνοντας τον αριθμό των hash functions παρατηρούμε ότι έχουμε διαφορετικές τιμές της Jaccard ομοιότητας του κειμένου 4 με τα υπόλοιπα πιο γειτονικά κείμενα. Δηλαδή ότι όσο αυξάνουμε το πλήθος των συναρτήσεων τόσο η ομοιότητα πέφτει αριθμητικά. Επιπλέον ότι κάνοντας το κ =5, αυτόματα περιορίζεται πάλι η ομοιότητα καθώς να βρεθεί ενα 5-single σε 2 κείμενα είναι λιγότερο πιθανό από ότι για κ=1,2,3,4,5.

Συμπερασματικά, όσο αυξάνουμε τον αριθμό των hash functions και κρατώντας το k σταθερό, παρατηρούμε ότι η Jaccard Ομοιότητα 'πέφτει' αριθμητικά, καθώς τα διαφορετικά shingles έχουν να κατανεμηθούν σε πολλά διαφορετικα hash buckets οπότε με την άυξηση αυτήν επιτυγχάνεται καλύτερος διαχωρισμός των hash values των singles, οπότε και μικρότερη ομοιότητα των κειμένων. Από την άλλη μερία καθώς αυξάνουμε το k, δηλαδή το πλήθος των λέξεων σε μια αλληλουχία(sequence), η Jaccard Ομοιότητα πάλι μειώνεται καθώς παρατηρείται ότι όσο αυξάνεται το k τα κείμενα φαίνονται όλο και πιο ανόμοια.

Επιπλέον υπάρχει το αρχείο write_clean_text.py το οποίο αποθηκεύει τα καθαρισμένα από special characters κείμενα στο cleaned_text.txt αρχείο, όπως ζητείται από την αναφορά.

Βιβλιογραφία

- **Mining of Massive Datasets**
- **Near Neighbor Search,**