Ένας τρόπος κατηγοριοποίησης κριτικών για ταινίες

Καρζής Αναστάσιος, Μανιάς Γεώργιος, Πηλιχός Χρήστος

Παρασκευή, 22 Φεβρουαρίου, 2019

Abstract

Στην παρούσα εργασία ενδιαφερόμαστε να επιτύχουμε την δημιουργία ενός μοντέλου μέσα από μηχανική μάθηση, το οποίο θα εκπαιδεύεται σε κριτικές ταινιών. Οι κριτικές είναι κατηγοριοποιημένες είτε ως θετικές, είτε ως αρνητικές. Για το σκοπό αυτό παρέχεται ένα σύνολο δεδομένων από ένα αρχείο .csv. Δοκιμάζουμε δύο διαφορετικούς αλγορίθμους κατηγοριοποίησης: Λογιστική Παλινδρόμηση (Logistic Regression) και Γραμμικό Κατηγοριοποιητή φορέα Δθιανυσμάτων (Linear SVC). Χρησιμοποιούμε τον διαχωρισμό των προτάσεων σε n-γράμματα. Επιλέγουμε τον καλλίτερο καταγεγραμμένο συνδυασμό παραμέτρων, τον οποίον και υλοποιούμε σε εφαρμογή.

Contents

1 Απαιτούμενες βιβλιοθήκες

Εν προκειμένω να υλοποιήσουμε την έρευνά μας, χρησιμοποιούνται οι κατωτέρω βιβλιοθήκες της Python οι οποίες παρέχουν τα απαραίτητα εργαλεία που επιθυμούμε να χρησιμοποιήσουμε στην εργασία.

In [1]:
```
# General purposes libraries
import pandas as pd
import re # for substring REplacement
import string # for string prossesing
import numpy as np # for calculations
import nltk # This is the library for preprocess functions
import nltk.classify.util
```

```
# Libraries from critics preprocess
from nltk import ngrams # Library to split a string into grams
# Library of finding the meaning root of a word in english language:
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer

# Classification algorithms for different approaches of critics interpretation
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from nltk.corpus import stopwords
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from nltk.classify import NaiveBayesClassifier

# Libraries for functions of classification algorithms' performance measurement
# Algorithms results' estimation measures
from sklearn.metrics import roc_curve, roc_auc_score, auc, accuracy_score
from sklearn.model_selection import train_test_split # Split a data set into a training set and a test set

import nltk
nltk.download('stopwords') # Get the stopwords list for the english language
nltk.download('punkt') # Get the punctuation symbols list
nltk.download('wordnet')

import time # For time measurement
```

Out[1]: True

2 Προεπεξεργασία των κριτικών

Όντας υποβεβλημένες από ανθρώπους (συνεπώς διαφορετικός τρόπος σκέψεις και σύνταξης) θα ήταν επιθυμητό οι κρικές να περιέλθουν σε μία κοινή μορφή ώστε να καταστεί δυνατή η σύγκρισή τους από έναν Η/Υ (ο οποίος και δεν αναγνωρίζει τα γλωσσικά ιδιώματα που χρησιμοποιούνται στον λόγο). Η ανωτέρω προεπεξεργασία αναπτύσσεται στην παρούσα ενότητα και διαρθρώνεται στα εξής βήματα: 1. Αφαίρεση λέξεων ίσσωνος πληροφορίας από τις κριτικές (Stopwords removal) 2. Αποβολή των καταλήξεων των λέξεων (Stemming) 3. Αναγωγή των λέξεων στη ρίζα τους (Lemmatize)

Γίνεται η υπόθεση πως κάθε συνάρτηση στην παρούσα ενότητα (το μέρος της προεπεξεργασίας) δέχεται ως είσοδό της ένα αντικείμενο της μορφής data frame. Με άλλα λόγια, το τύπο τον οποίον η Python αποθηκεύει τα δεδομένα όταν τα διαβάζει από ένα .csv αρχείο.

2.1 Αφαίρεση προσωρινών λέξεων-κλειδιά

Πρωτίστως, ορίζεται ένα σύνολο "απαγορευμένων λέξων" οι οποίες θα ήταν επιθυμητό να εξαιρεθούν όποτε απαντώνται στις κριτικές, τις οποίες και καλούμε **προσωρινές λέξεις-κλειδία** (stopwords). Ο λόγος είναι πως οι προκείμενες λέξεις είναι απαραίτητες για την επικοινωνία ανάμεσα σε ανθρώπους. Έτσι λειτουργούν ως βοηθητικές λέξεις ώστε ένα ανθρώπινο ον να μπορέσει να εξάγει το επιθυμητό συμπέρασμα. Ούσες βοηθητικές, απαντώνται συχνά μέα στις κριτικές. Ο Η/Υ ο οποίος δεν γνωρίζει του κανόνες που διέπουν την ανθρώπινη επικοινωνία να τις λάβει υπ' όψιν του, έχοντας ως συνέπεια * αφ' ενός το σύνολο των διαφορετικών λέξεων που καλούμαστε να χειριστούμε (:λεξικό) να μεγαλώσει * αφ' ετέρου οι λέξεις αυτέ δεν προσδίδουν κάποια πληροφορία για την κριτική της ταινίας, παρά μόνον

διευκολύνουν την ανθρώπινη επικοινωνία * έτσι ακόμη και δύο κριτικές με το ίδιο νόημα διαφέρουν καθ' ότι είναι συνταγεμένες από διαφορετικούς χρήστες.

Κατά συνέπεια οι λέξεις αυτές δεν προσφέρουν κάποιο κέρδος πληροφορίας και παραπλανούν τον Η/Υ από το ζητούμενο: την απόφανση εάν η κριτική είναι θετική ή αρνητική.

Η λίστα με των ανωτέρω λέξεων που επιθυμούμε να εξαιρεθούν από τις κριτικές μπορεί να δημιουργηθεί: * είτε με ρητή δήλωση * είτε χρησιμοποιώντας κάποια έτοιμη λίστα.

Επιλέγουμε τη δεύτερη δυνατότητα. Έτσι, ούσα η αγγλική η γλώσσα που είναι γραμμένες οι κριτικές) ανακτούμε τη λίστα με τις βοηθητικές λέξεις από τη βιβλιοθήκη `nltk`.

```
In [2]: nltk.download('stopwords')
        Stop_words = set(stopwords.words('english'))
```

Η συνάρτηση για την αφαίρεση των προσωρινών λέξεων-κλειδιά προσπελάβνει μία γραμμή του αρχείου (ενδέχεται να είναι μια καλή ή μια κακή κριτική), την αποθηκεύει στη μεταβλητή "protasi" και διενεργεί ως εξής: 1. Η μεταβλητή "protasi" ούσα αλφαριθμητική ακολουθία: 1. διαχωρίζεται σε μέρη -τις λέξεις της φυσικής που την αποτελούν- με τη συνάρτηση `split()` και 2. ελέγχεται εάν το κάθε μέρος της -το αποθηκεύουμε στη μεταβλητή "word"- είναι μία stopword. 2. Η αρχικώς αναγνωσμένη αλφαριθμητική ακολουθία, αντικαθίσταται από το αποτέλεσμα της παραπάνω διεργασίας, ήτοι μία αλφαριθμητική ακολουθία απηλλαγμένη από stopwords. 3. Εφαρμόζεται η ανωτέρω διαδικασία σε κάθε διαθέσιμη κριτική. 4. Το αποτέλεσμα αποθηκεύεται υπό μορφή data frame.

Η αλγοριθμική υλοποίηση των ανωτέρω έγκειται στην εξής συνάρτηση:

```
In [3]: def remove_Stopwords(x):
            cur_ln = 0 # Let's start from the beggining
            for protasi in x:
                    # Convert to lower case and get every word:
                protasi = [word for word in protasi.lower().split() if word not in Stop_words]
                protasi = ' '.join(protasi) # We need to initialize the resutl with the empty string ' '
                # Replace the original data, with the stopword free senetence in the data frame:
                x.loc[cur_ln] = protasi
                cur_ln+=1 # Move to the next line
            return(x) # Return the modified (ie stopwords free) data frame
```

2.2 Αφαίρεση σημείων στίξης

Άλλη μια βοηθητική λειτουργία της ανθρώπινης γλώσσας -και δη του γραπτού λόγου- είναι η χρήση των σημείων στίξης. Στη πληθώρα των περιπτώσεων -όσον τη σκοπιά του Η/Υ- τα σημεία στίξης είναι αδιάφορα (ένας λόγος είναι ότι οι λέξεις "γεια" και "γεια!" λογίζονται ως διαφορετικές). Ωστόσο, το θαυμαστικό επί παραδείγματι θα μπορούσε να δηλώσει την υπέρμετρη υποστήριξη της θετικής/αρνητικής άποψης του χρήστη, ή το ερωτηματικό την αμφιβολία του για τα λεγόμενα του. Εάν και τα σημεία στίξεως μπορούν να προσδόσουν μία (θετικά ή αρνητικά) βεβαρυμένη αξιολόγηση των ειπομένων, είναι μία διεργασία η οποία αφήνεται για μελλοντική βελτίωση του μοντέλου.

Πίσω στη θεώρησή μας για την αφαίρεση των σημείων στίξεως, κάθε γραμμή εξετάζεται ξεχωριστά και διενεργόνται τα εξής:

1. Διαγραφή των σημειών στίξεως πλην της τελείας και του ερωτηματικού

2. Αντικατάσταση της τελείας και του ερωτηματικού με λευκό χώρο (ήτοι τον κενό χαρακτήρα).

Ο λόγος που διατηρούνται οι τελείες και τα κόμματα σε πρώτο χρόνο και αντικαθίστανται με κενούς χαρακτήρες εν δευτέροις είναι επειδή τα εν λόγω σημεία στίξεως σηματοδοτούν το πέρας μιας πρότασης (στη φυσική γλώσσα), ήτοι εμφανίζονται (κατά πάσα πιθανότητα) στο τέλος μιας νοηματικής περιόδου. Όλα τα άλλα σημεία στίξεως εμφανίζονται εμβόλημα στην νοηματική περιόδο (για παράδειγμα το "_"

μπορεί να σημαίνει τη συνένωση δύο λέξεων σε μία διεύθυνση ηλεκτρονικού ταχυδρομείου) και δεν συνεισφέρουν σε πληροφορία. Τελικά, το εξαγόμενο είναι ένα data frame με εγγραφές απηλλαγμένες από σημεία στίξεως.

```
In [4]: def remove_Stiksis(x):
            cur_ln = 0 # Start from the first sentence
            for protasi in x: # For every line (ie a whole critic)...
                cleanr = re.compile('<.*?>')
                protasi = re.sub(r'\d+', '', protasi)
                protasi = re.sub(cleanr, '', protasi)
                protasi = re.sub("'", '', protasi) # Remove single quotes (') from protasi
                # \W+ option splits protasi into a list with words
                # \(ie parts in between white spaces) ignoring pucntuation marks:
                protasi = re.sub(r'\W+', ' ', protasi)
                protasi = protasi.replace('_','') # Remove underscores (_) from protasi
                x.iloc[cur_ln]=protasi # Replace the processed line with the modified (ie punctuation free) one
                cur_ln += 1 # Move to the next line (ie sentence, ie critic)
            return(x) # Return the modified (ie punctuation free) data frame
```

2.3 Λημματοποίηση των κριτικών

Η **λημματοποίηση** (lemmatisation) μιας λέξεως είναι η διαδικασία κατά την οποία βρίσκεται η συγγενής έννοια η οποία απαντάται στα λεξικά (ο όρος ενός λεξικού ή μιας εγκυκλοπαίδειας καλείται λήμμα, εξ ου και τ' όνομα της διαδικασίας). Συνεπώς, κάθε λέξη των κριτικών ανάγεται σε μια **κανονική μορφή**, ήτοι μια μονοσήμαντα ορισμένη λέξη, νοηματικά ταυτόσημη και με την απλούστερη δυνατή μορφή (όπως ονομαστική πτώση ενικού αριθμού (εάν είναι ουσιαστικό) στο αρσενικό γένος (εάν είναι επίθετο) ή α' ενικό πρόσωπο ενεστώτα (για τα ρήματα), ...). Η προκειμένη κανονική μορφή είναι η νοηματική της ρίζα, δηλαδή στο λήμμα με το οποίο είναι καταγεγραμμένη σε ένα λεξικό ή μια εγκυκλοπαίδεια. Εδώ η λημματοποίηση μιας λέξεως θα αποδόσει μία λέξη των αρχαίων αγγλικών (η οποία έχει μετεξελιχθεί δια μέσω των αιώνων ώστε να περιέλθει στη σημερινή μορφή που χρησιμοποιείται).

Κατά τη θεώρησή μας χρησιμοποιείται η συνάρτηση `WordNetLemmatizer()` της βιβλιοθήκης `nltk` κατά τον κάτωθι τρόπο σε κάθε γραμμή του παρεχόμενου data frame: 1. Εύρεση των λέξεων που αποτελούν τη γραμμή (πρόταση) ως την αλφαριθμητική υπακολουθία που περιέχεται ανάμεσα σε δύο διαδοχικούς κενούς χαρακτήρες 2. Για κάθε μία από τις λέξεις τις πρότασης: 1. Η λέξη ανάγεται στο λήμμα της (ήτοι στην "κανονική μορφή" της) 2. Το αποτέλεσμα αποθηκεύεται σε έναν buffer τον οποίον καλούμε riza (και είναι ένας μονοαδιάστατος πίνακας). 3. Όταν έχουν προσπελαστεί άπασες οι λέξεις που συναποτελούν την πρόταση (:κριτή/γραμμή του data frame) το αποτέλεσμα που βρίσκεται στο διάνυσμα riza, αντικαθιστάει τη γραμμή που προσπελάστηκε.

Σε όρους της γλώσσας Python, η παραπάνω μεθοδολογία υλοποιείται όπως κατωτέρω:

```
In [5]: def rizes(x):
            arxiki = WordNetLemmatizer()

            cur_ln = 0 # Start from the very first line of the data frame
            riza = [] # Initialize to an empty data frame the buffer to which we append the lemmatized words
            for protasi in x:
            # Creates a list which elements are tokens (ie protasi's substrings which are included in between two consequ
                protasi_list = word_tokenize(protasi)
                for leksi in protasi_list:
                    riza.append(arxiki.lemmatize(leksi)) # Append to the buffer riza the current lemmatized word
                protasi = ' '.join(riza) # Store the lemmatized result to a variable called: protasi
                # Replace the current entry (.iloc[] function is used to call the data by row number) with the modified
```

```
        x.iloc[cur_ln] = protasi
        cur_ln += 1 # Move to the next line
        riza = [] # Reinitialize to the empty data frame
    return(x) # Return the data frame with the lemmatized entries
```

2.4 Αγνόηση των καταλήξεων των λέξεων

Επειδή υπάρχουν αρκετά και διαφορετικά μέρη του λόγου, η αναγωγή σε λήμματα μόνον δεν αρκεί. Έτσι τα λήμματα, όντας λέξεις, διακρίνονται σε ουσιαστικά, επίθετα, ρήματα, επιρρήματα. Καθ' ένα εξ αυτών δικρίνεται στο θέμα το οποίο ακολουθείται από μια κατάληξη. Αναλόγως, το μέρους του λόγου υπάρχουν διαφορετικές καταλήξεις (στα ελληνικά, τα επιρήμματα τελειώνουν σε -ως ή -α, τα επίθετα σε -ος, -η, -ο, τα ρήματα -ω, -όμαι και διάφορες άλλες καταλήξεις αναλόγως της φωνής/διάθεσης/χρόνου). Έτσι θα ήταν επιθυμητό να απαλλαχθούν οι λέξεις από τις καταλήξεις τους, έτσι ώστε μέρη του λόγου με ίδια σημασιολογική ερμηνεία να ταυτιστούν.

Η αλγοριθμική υλοποίηση για την αγνόηση των καταλήξεων στις λέξεις συνίσταται στην εφαρμογή στη κάθε γραμμή (κριτική) ενός data frame των εξής βημάτων: 1. Ανακτώνται οι λέξεις της κάθε πρότασης. 2. Με χρήση της συνάρτησης `SnowballStemmer("english")` αφαιρούνται οι καταλήξεις από τη κάθε μία λέξη της πρότασης. 3. Η ελευθέρα καταλήξεων λέξη προσαρτάται στον buffer "riza". 4. Όταν θα έχουν προσπελαστεί όλες οι λέξεις, το αποτέλεσμα του buffer αντικαθιστά την τρέχουσα προσπελαύνουσα γραμμή. 5. Η συνάρτηση επιστρέφει το data frame με τα στοιχεία των γραμμών του ελεύθερα καταλήξεων.

```
In [6]: def no_katalikseis(x):
            syllabopoihsh = SnowballStemmer("english")

            cur_ln = 0 # Start from the very first line
            riza = [] # Initialize the buffer to the empty data frame
            for protasi in x:
                # Creates a list which elements are tokens (ie protasi's substrings which are included
                # in between two consequtive white space characters)
                protasi_list = word_tokenize(protasi)
                for leksi in protasi_list:
                    riza.append(syllabopoihsh.stem(leksi)) # Drop the endings from the words
                protasi = ' '.join(riza) # Join the ending free word leaving
                # Replace the result to the line processed (which line is called b its row index via .iloc[] function
                x.iloc[cur_ln] = protasi
                cur_ln += 1 # Move to the next line
                riza = [] # Re-initialize the buffer to the empty data frame
            return(x) # Return the data frame with ending free critics
```

3 Θεωρία για το κύριο μέρος του αλγορίθμου

Το σύνολο δεδομένων αποτελείται από 100000 γραμμές με 5 γνωρίσματα, τα: 1. id: ο αύξων αριθμός της κριτικής 2. type: το οποίο υποδεικνύει εάν η εν λόγω κριτική θα συγκαταλεγεί στο σύνολο εκπαίδευσης ή στο σύνολο δοκιμής. 3. review: προκειται για το σώμα της κριτικής, ήτοι τη γνώμη του χρήστη για τη ταινία. 4. label: απόφανση εάν η κριτική διάκειται υπέρ (pos), κατά της ταινίας (neg) ή δεν τυγχάνει κατηγοριοποίησης (unsup) σε θετική ή αρνητική. 5. file: είναι η πηγή (αρχείο) απ' όπου αντλήθηκε η κριτική.

3.1 Καθαρισμός των κριτικών

Αρχικώς, αναγινώσκονται τα δεδομένα και απορρίπτονται οι κριτικές οι οποίες δεν έχουν κατηγοριοποιηθεί (στήλη label) ως θετικές (label=pos) ή ως αρνητικές (label=neg). Κατ' επέκτασην οι κριτικές που δεν τυχάνουν θετικής ή αρνητικής υφής (label=unsup) απορρίπτονται από τη συνέχεια της ερεύνης μας.

```
In [7]: ## main function
        data = pd.read_csv("imdb_master.csv", delimiter = ',', encoding ="latin-1")
        data = data.drop(['Unnamed: 0', 'file'], axis = 1)
        data = data[:][data.label.isin(['pos','neg'])].reset_index(drop = True)
```

Κατ' όπιν ανακτώνται οι εγγραφές (γραμμές) του συνόλου δεδομένων που αποτελούν το σύνολο δοκιμής (type=train) και το σύνολο δοκιμής (type=test). Εφ' όσον πρόκειται να αποθηκευθούν σε ξεχωριστή μεταβλητή μόνον κριτικές για την εκπαίδευση του μοντέλου (type=train), διατηρούμε μόνο το σώμα της κριτικής (στήλη review) και προς τα που είναι κείμενη (στήλη label).

```
In [8]: training_data = data[["review", "label"]][data.type.isin(['train'])].reset_index(drop=True)
        test_data = data[["review", "label"]][data.type.isin(['test'])].reset_index(drop=True)
        print(training_data.head())
```

	review	label
0	Story of a man who has unnatural feelings for …	neg
1	Airport '77 starts as a brand new luxury 747 p…	neg
2	This film lacked something I couldn't put my f…	neg
3	Sorry everyone,,, I know this is supposed to b…	neg
4	When I was little my parents took me along to …	neg

Για αλγοριθμικούς λόγους προτιμούμε τις αριθμητικές τιμές έναντι των αλφαριθμητικών γνωρισμάτων. Έτσι στη στήλη label εφαρμόζεται ο μετασχηματισμός $\begin{cases} pos \longmapsto 1 \\ neg \longmapsto 0 \end{cases}$.

```
In [9]: training_data[["label"]] = np.where(training_data[["label"]] == "pos", 1, 0)
        test_data[["label"]] = np.where(training_data[["label"]] == "pos", 1, 0)
        print(training_data.head())
```

	review	label
0	Story of a man who has unnatural feelings for …	0
1	Airport '77 starts as a brand new luxury 747 p…	0
2	This film lacked something I couldn't put my f…	0
3	Sorry everyone,,, I know this is supposed to b…	0
4	When I was little my parents took me along to …	0

Διατηρούμε τη στήλη label ως τη στήλη στόχο, ήτοι τις πργματικές τιμές για την κλάση όπου ανήκουν οι κριτικές του συνόλου εκπαίδευσης (train).

```
In [10]: target_train = training_data[["label"]].values.tolist()
         target_test = test_data[["label"]].values.tolist()
```

Αφού πλέον η στήλη στόχος (label) με τις πραγματικές τιμές (1 ή 0) των κλάσεων (pos ή neg) έχει αποθηκευθεί σε ξεχωριστή μεταβλητή, διαγράφεται από το σύνολο εκπαίδευσης.

```
In [11]: training_data = data[["review"]][data.type.isin(['train'])].reset_index(drop=True)
         test_data = data[["review"]][data.type.isin(['test'])].reset_index(drop=True)
```

Τέλος, οι κριτικές περιέρχονται στην επιθυμητή μορφή, σύμφωνα με τη θεωρία πυο αναπτύχθηκε στην πρότερη ενότητα.

```
In [12]: training_data['review'] = no_katalikseis(rizes(remove_Stopwords(remove_Stiksis(training_data['review']))
         test_data['review'] = no_katalikseis(rizes(remove_Stopwords(remove_Stiksis(test_data['review']))))
         print(training_data.head())
```

```
                                              review
0  stori man unnatur feel pig start open scene te…
1  airport start brand new luxuri plane load valu…
2  film lack someth couldnt put finger first char…
3  sorri everyon know suppos art film wow hand gu…
4  littl parent took along theater see interior o…
```

3.2 tfIdf

Το TfIdf αποτελεί ένα στατιστικό μέτρο που χρησιμοποιείται για την αξιολόγηση της σημασίας μιας λέξης σε ένα έγγραφο μιας συλλογής ή ενός συνόλου κειμένων. Ορίζεται

$$\text{tfIdf}(t) := \frac{\text{συχνότητα του όρου t (tf: term frequency)}}{\text{αντίστροφη συχνότητα εγγράφων (idf: inverse document frequency)}}$$

Η σημασία αυξάνεται αναλογικά με τον αριθμό των φορών που εμφανίζεται μια λέξη σε ένα κείμενο, αλλά αντισταθμίζεται από τη συχνότητα εμφάνισης της λέξης εντός της συλλογής ή του συνόλου κειμένων που αναλύουμε.

Αναλυτικότερα, έχουμε

$$\text{tf}(t) := \frac{\text{αριθμός εμφανίσεων του όρου } t \text{ εντός του εγγράφου}}{\text{συνολικό πλήθος όρων}}$$

$$\text{idf}(t) := \frac{\log(\text{πλήθος των εγγράφων στη συλλογή})}{\text{το πλήθος των εγγράφων όπου απαντάται ο όρος } t}$$

3.3 Δημιουργία συνόλου εκπαίδευσης και δοκιμής

Για τον διαχωρισμό των κριτικών που εμπεριέχονται στο σύνολο εκπαίδευσης χρησιμοποιούμε τη συνάρτηση

$$X\_train,\ X\_val,\ y\_train,\ y\_val \leftarrow \texttt{train\_test\_split}(X,\ target,\ train\_size = 0.75)$$

η οποία επιστρέφει τυχαία επιλεγμένα στοιχεία από το data frame X στο αποτέλεσμά της * X_train όσον αφορά το σύνολο εκπαίδευσης και X_val όσο αφορά το σύνολο δοκιμής και * τις γραμμές του data frame ως y_train και y_val στις οποίες αντιστοιχεί το κάθε στοιχείο του X_train και y_train αντίστοιχα.

Έτσι $\text{y\_train} \cap \text{y\_val} = \varnothing$, $|\text{y\_train}| + |\text{y\_val}| = |X|$ και $\frac{|\text{X\_train}|}{|X|} \approx 0.75$.

3.4 n-γράμματα

Έστω $n \in \mathbb{N}$. Ως n-γραμμα θεωρείται ο διαχωρισμός μιας αλφαριθμητικής ακολουθίας σε όλες τις δυνατές n-άδες συνεχόμενων λέξεων[1]. Έτσι αν και αυξάνεται το λεξιλόγιο (κι άρα η υπολογιστική πολυπλοκότητα του αλγορίθμου μας) προσπαθούμε να συγκαταλέξουμε εκφράσεις της μορφής "not so good" στις αρνητικές. Εάν δεν χρησιμοποιούνταν τα n-γράμματα (ή ισοδύναμα να χησιμοποιούνταν τα 1-γράμματα), τότε η έκφραση "not so good", θα λογιζόταν ως 3 ξεχωριστά στοιχεία: "not", "a", "good" και η κριτική θα συγκαταλεγόταν στις θετικές (χάριν του "good").

[1] Μια υπακολουθία ανάμεσα σε δύο διαδοχικούς λευκούς χαρακτήρες.

Για λόγους υπολογιστικών πόρων, αλλά και πως δεν θεωρούμε πως ανάμεσα στο αρνητικό μόριο "not" και στην αποτίμηση "good" μεσολαβούν περισσότερες των μία λέξεων (επί παραδείγματι, η κριτική "not such a good" δεν θα συγκαταλεγεί ποτέ στις αρνητικές, καθ' ότι στις λέξεις "not" και "good" παρεμβάλλονται υπεράνω της μία λέξεων).

4 Κυρίως μέρος της συνάρτησης

Διατηρούμε μόνον τα σώματα των κριτικών (ήτοι τις απόψεις των χρηστών):

```
In [13]: df = pd.DataFrame(training_data[["review"]])
         training_data = df["review"].tolist()

         df_test = pd.DataFrame(test_data[["review"]])
         test_data = df_test["review"].tolist()
```

Κατ' όπιν εφαρμόζουμε τους αλγορίθμους μας για να ελέγξουμε την απόδοσή τους:

```
In [14]: #Try for 1,2,3-grams
         for n in range(1,4):
                                #Change strings into vectors (vectorization):
             tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,n))
             tfidf_vectorizer.fit(training_data)
             X = tfidf_vectorizer.transform(training_data)
             X_test = tfidf_vectorizer.transform(test_data)

             #Split the training data in training and evaluating data for algorithms
             X_train, X_val, y_train, y_val = train_test_split(X, target_train, train_size = 0.75)

             # Try for C = 0.5, 1, 1.5, 2 for Logistic Regression algoritmh
             for i in range(1,4):
                 start = time.time()
                 lr = LogisticRegression(C=i*0.5)
                                    Traing the model on the splitted training data set
                 lr.fit(X_train, y_train)
                 print("LogReg for "+ str(n)+"-gramms and for C=" + str(i*0.5)
                     +": Accuracy = ", round((accuracy_score(y_val, lr.predict(X_val)) * 100), 2))
                                    Test model's performance on the (real) test set
                 final_tfidf = LogisticRegression(C=i*0.5)
                 final_tfidf.fit(X, target)
                 end = time.time()
                 dif = round((end - start),2)
                 print ("LogReg for "+ str(n)+"-gramms and for C=" + str(i*0.5) +": Train Time: "+ str(dif)
                +"s. Final Accuracy: ", round((accuracy_score(target_test, final_tfidf.predict(X_test))*100),2))

             # Try for C = 0.5, 1, 1.5, 2, for Support Vector Classifier
             for i in range(1,4):
                 start = time.time()
                                    # Train the model on the splitted training set
                 svc = LinearSVC(C=i*0.5)
                 svc.fit(X_train, y_train)
                 print ("SVC for "+ str(n)+"-gramms and for C=" + str(i*0.5)
                     + ": Accuracy: ", round((accuracy_score(y_val, svm.predict(X_val)) * 100),2))
```

```
                                    # Test algorithm's performance on the (real) test set
                final_svc_ngram = LinearSVC(C=i*0.5)
                final_svc_ngram.fit(X, target)
                end = time.time()
                dif = round((end - start),2)
                print ("SVC for "+ str(n)+"-gramms and for C=" + str(i*0.5) +": Train Time:"+ str(dif)
            +"s. Final Accuracy: ", round((accuracy_score(target_test, final_svc_ngram.predict(X_test))*100),2
```

```
For 1-gramms and for C=0.5: Accuracy = 88.26
For 1-gramms and for C=0.5: Train Time: 1.18s. Final Accuracy: 87.7
For 1-gramms and for C=1.0: Accuracy = 88.59
For 1-gramms and for C=1.0: Train Time: 1.58s. Final Accuracy: 87.79
For 1-gramms and for C=1.5: Accuracy = 88.88
For 1-gramms and for C=1.5: Train Time: 1.36s. Final Accuracy: 87.89
For 1-gramms and for C=0.5: Accuracy: 88.82
For 1-gramms and for C=0.5: Train Time:0.74s. Final Accuracy: 87.27
For 1-gramms and for C=1.0: Accuracy: 88.35
For 1-gramms and for C=1.0: Train Time:0.76s. Final Accuracy: 86.4
For 1-gramms and for C=1.5: Accuracy: 88.03
For 1-gramms and for C=1.5: Train Time:1.11s. Final Accuracy: 85.82
For 2-gramms and for C=0.5: Accuracy = 86.75
For 2-gramms and for C=0.5: Train Time: 3.81s. Final Accuracy: 87.16
For 2-gramms and for C=1.0: Accuracy = 87.41
For 2-gramms and for C=1.0: Train Time: 3.94s. Final Accuracy: 87.9
For 2-gramms and for C=1.5: Accuracy = 87.97
For 2-gramms and for C=1.5: Train Time: 5.11s. Final Accuracy: 88.24
For 2-gramms and for C=0.5: Accuracy: 88.82
For 2-gramms and for C=0.5: Train Time:1.85s. Final Accuracy: 88.95
For 2-gramms and for C=1.0: Accuracy: 89.06
For 2-gramms and for C=1.0: Train Time:2.46s. Final Accuracy: 88.91
For 2-gramms and for C=1.5: Accuracy: 89.22
For 2-gramms and for C=1.5: Train Time:3.39s. Final Accuracy: 88.96
For 3-gramms and for C=0.5: Accuracy = 86.9
For 3-gramms and for C=0.5: Train Time: 5.66s. Final Accuracy: 86.35
For 3-gramms and for C=1.0: Accuracy = 87.58
For 3-gramms and for C=1.0: Train Time: 6.8s. Final Accuracy: 87.14
For 3-gramms and for C=1.5: Accuracy = 88.14
For 3-gramms and for C=1.5: Train Time: 7.54s. Final Accuracy: 87.61
For 3-gramms and for C=0.5: Accuracy: 89.46
For 3-gramms and for C=0.5: Train Time:2.97s. Final Accuracy: 88.71
For 3-gramms and for C=1.0: Accuracy: 89.86
For 3-gramms and for C=1.0: Train Time:4.3s. Final Accuracy: 88.94
For 3-gramms and for C=1.5: Accuracy: 89.97
For 3-gramms and for C=1.5: Train Time:5.76s. Final Accuracy: 89.06
```

Συμπέρασμα. *Από τα ανωτέρω αποτελέσματα επιλέγουμε τον Γραμμικό Κατηγριοποιητή Φορέα Διανυσμάτων (SVC) για παράμετρο* $C = 1.5$ *και* 3*-γράμματα.*