ΔΙΠΛΩΜΑΤΙΚΗ Ε Ρ Γ Α Σ Ι Α

Σπουδαστές:

Νάτσης Αινείας
Τρούλης Μιχάλης

Θέμα:

“ Μελέτη και ανάπτυξη διαδικτυακής εφαρμογής, με βάση κειμενική πληροφορία και στόχο την εξαγωγή συναισθήματος από αυτό.”

Εισηγητής:

Παπαδάκης Νικόλαος

Περίληψη

Η παρούσα πτυχιακή εργασία πραγματεύεται το αντικείμενο της συναισθηματικής ανάλυσης σε ηλεκτρονικής μορφής έγγραφα, χρησιμοποιώντας κατηγοριοποιητές κειμένων είτε βασισμένους σε λεξικά αλλά και Naive Bayes οι οποίοι βασίζονται στην εκπαίδευση απο άλλα κείμενα. Για να γίνει επαλήθευση των αποτελεσμάτων του αλγορίθμου που εξήγαγε τα καλύτερα αποτελέσματα θεωρήθηκε σκόπιμη η δημιουργία μιας διαδικτυακής εφαρμογής στην οποία οι χρήστες συνδέονται και αξιολογούν με την σειρά τους τα ίδια κείμενα. Η εφαρμογή συγκρίνει τα αποτελέσματα του χρήστη με του αλγορίθμου και βγάζει στο ποσοστό επιτυχίας του.

Abstract

The following thesis deals with the object of sentiment analysis of electronic textual information, with the use of text categorisation based on dictionaries and Naive Bayes algorithms, which are trained with other texts. To evaluate the best algorithm, was considered the creation of a web application in which the users sign in and evaluate the same textual information. The application compares the user's and the algorithm's results and results the algorithm's percentage of success.

Table of Contents

Κατάλογος εικόνων

Κατάλογος Πινάκων

1 ΚΕΦΑΛΑΙΟ: Εισαγωγή

1.1 Πρόλογος

Η έμφυτη ανάγκη του ανθρώπου να αποκτά αλλά και να συλλέγει την γνώση έχει οδηγήσει στην αύξηση εγγράφων κειμένων που δημοσιεύονται καθημερινά στα μέσα κοινωνικής δικτύωσης αλλά και σε όλες τις ιστοσελίδες. Αυτό όμως που έχει περισσότερο ενδιαφέρον για τον σύγχρονο άνθρωπο είναι η εύρεση της κοινής γνώμης για όλα τα κοινωνικά ζητήματα, αλλα και για τον ίδιο εφοσων πρόκειται για δημόσιο και γνωστό πρόσωπο.

Η αλματώδης αύξηση των εγγράφων καθιστά δύσκολη την συλλογή χρήσιμης πληροφορίας καθώς υπάρχει πάντα και η άχρηστη πληροφορία η ο θόρυβος. Παρόλα αυτά όλος ο όγκος της πληροφορίας περιέχει επαρκή δεδομένα και μπορεί να χρησιμοποιηθεί για να διαχωρίσουμε την χρήσιμη για εμάς πληροφορία από τον θόρυβο.

Οι τεχικές συναισθηματικής ανάλυσης ανήκουν σε τρεις κυρίως κλάδους, αυτόν της τεχνητής νοημοσύνης και μηχανικής μάθησης, της υπολογιστικής γλωσσολογίας (computational linguistics) και της εξόρυξης κειμένου (text mining). Μέσω του ευρύτερου κλάδου της τεχνητής νοϋμοσύνης στον οποίο συγκαταλεγονται οι τεχνικές της εξόρυξης γνώσης από δεδομένα (data mining), η εξόρυξη κειμένου (text mining) και η μηχανική μάθηση (machine learning), επιτυγχάνουμε την κατανόηση κειμένων όπως και την συλλογή της επιθυμητής πληροφορίας.

Η εργασία αυτή πραγματεύεται την ανάλυση κειμένων ως προς το συναίσθημα του περιεχόμενού τους (sentiment analysis) και στην συνέχεια παρουσιάζει το τελικό εργαλείο που αξιολογεί (μέσω της ανθρώπινης παρουσίας) τις βιβλιοθήκες που χρησιμοποιήσαμε sentiment detection classifiers. Στην παρούσα πτυχιακή εργασία οι τεχνικές που χρησιμοποιήθηκαν είναι η εξώρυξη κειμένου και η μηχανική μάθηση. Για την ανάπτυξη του λογισμικού χρησιμοποιήθηκε η γλώσσα Ruby και για την εφαρμογή αξιολόγισης το framework Ruby On Rails 4.0.0 το οποίο χρησιμοποιείται για την ανάπτυξη ιστοσελίδων.

1.2 Δομή

Η εργασία αποτελείται απο έξι κεφάλαια συνολικά. Αυτό είναι το πρώτο κεφάλαιο στο οποίο γίνεται αναφορά στο γενικό περιεχόμενο της, αλλά και στο πώς είναι διαρθρωμένα τα κεφάλαια.

Στο δεύτερο κεφάλαιο εξηγούνται οι ορολογίες των τεχνικών μηχανικής μάθησης που εφαρμόστηκαν για να ακολουθήσουν στην συνέχεια όλες οι βιβλιοθήκες που χρησιμοποιήθηκαν καθώς και τα αποτελέσματα που εξήγαγαν. Αυτό αποτελεί και το τρίτο κεφάλαιο της εργασίας στο οποίο επιχειρείται η καταγραφή των πειραμάτων που διεξήχθησαν απο τις βιβλιοθήκες που χρησιμοποιήθηκαν με σκοπό την σύγκριση των αποδόσεων τους και την επιλογή της βιβλιοθήκης με τα πιο ικανοποιητικά αποτελέσματα. Για την αξιολόγιση των βιβλιοθηκών έπρεπε να συλλεχθούν κείμενα τα οποία παρουσιάζονται στο ίδιο κεφάλαιο όπως και τα προβλήματα που

αντιμετωπίστηκαν και τις λύσεις που δώθηκαν σε αυτά.

Στο τέταρτο κεφάλαιο καταγράφονται τα μερη απο τα οποία απαρτίζεται η διαδικτυακή εφαρμογή, οι αρχιτεκτονικές και τα είδη των τεχνολογιών που χρησιμοποιήθηκαν κατα την διαρκεια της κατασκευής της.

Στο πέμπτο και τελευταίο κεφάλαιο εξηγούνται ολές οι λειτουργίες της εφαρμογής με την αναλυτική παρουσίαση εικόνων και πως συνεργάζονται για να παρέχουν στον χρήστη το τελικό περιβάλλον. Ακολουθεί και μια αναφορά σε όλα τα σχεδιαστικά προβλήματα και διλήμματα που αντιμετωπίστηκαν και την τελική λύση αυτών, καθώς και τι μελλοντικές αλλαγές και προσθήκες μπορούν να γίνουν στην εργασία.

2 ΚΕΦΑΛΑΙΟ: Εξόρυξη Γνώσης- Γνώμης (opinion mining)

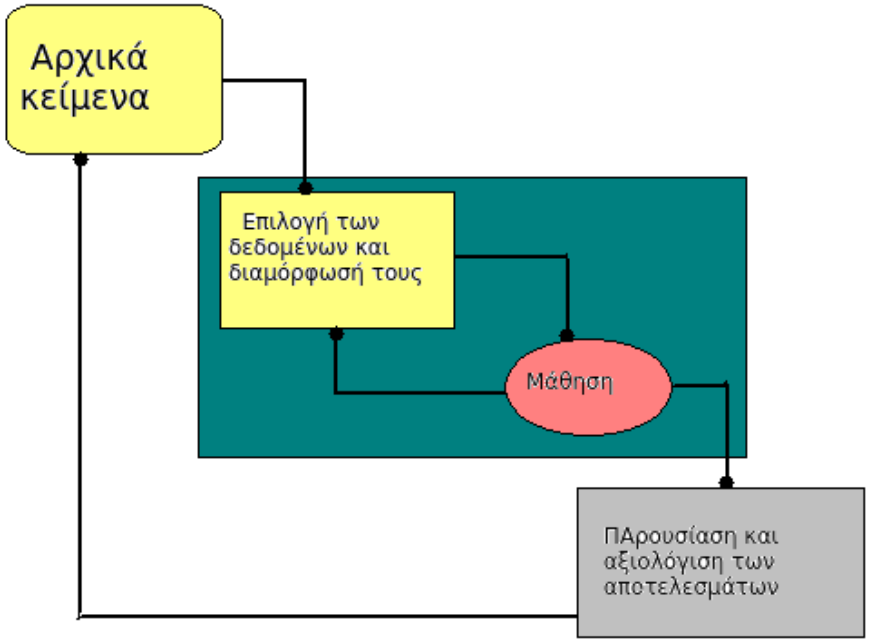
Η εξαγωγή συναισθημάτων από κείμενα διαφορετικής θεματολογίας είναι ένα σημαντκό παιδίο έρευνας και τείνει να γίνει ακόμα μεγαλύτερο με την εισχώρηση της ένοιας big data στην επιστήμη των υπολογιστών. Ο γενικότερος όρος που χρησιμοποιείται και έχει γίνει γενικά αποδεκτός ειναι ή Εξόρυξη Γνώμης (Opinion Mining), ο οποίος δεν περικλείει την συναισθηματική ανάλυση, αλλά αποτελεί υποκατηγορία της Εξόρυξης Δεδομένων (Data mining).

Μπορούμε να ορίσουμε την κειμενική εξόρυξη γνώσης (text mining) ώς μια αυτοματοποιημένη διαδικασία ανακάλυψης άγνωστων πληροφοριών που προέρχονται απο διαφορετικές κειμενικές πηγές. Για να καταλήξουμε στην διαδικασία της εξόρυξης πρέπει αρχικά να έχει προεπεξεργαστεί το κείμενο (εξάλειψη τυχών κειμενικών ιδιαιτεροτήτων, αναγραματισμών και αποθήκευση των τελικών δεδομένων σε μια βάση δεδομένων) να εφαρμοστεί το μοντέλο επεξεργασίας όπως η μηχανική μάθηση, και τελικά να αξιολογηθούν τα τελικά αποτελέσματα.

Αναφέρουμε μερικές απο τις κατηγορίες των μεθόδων που χειρίζεται η εξόρυξη κειμένου:

- Γλωσσικός προσδιορισμός (Language Identification)
- Εξαγωγή χαρακτηριστικών γνωρισμάτων (Feature Extraction)
- Περιληπτική Παρουσίαση της Πληροφορίας (Summarization)
- Κατηγοριοποίηση, κατάταξη με επίβλεψη (Categorization, Supervised Classification)
- Ομαδοποίηση, μη επιβλεπόμενη κατάταξη (Clustering, Unsupervised Classification)

Δεν πρέπει να συγχέεται ο όρος text mining με την απλή αναζήτηση στον παγκόσμιο ιστό (google). Με την απλή αναζήτηση ο χρήστης ψάχνει να βρεί μια πληροφορία η οποία υπάρχει σιγουρα στον ιστό, σε αντίθεση με εξόρυξη γνώσης η οποία έχει ώς στόχο να ανακαλύψει άγνωστη πληροφορία χρησιμοποιώντας την ήδη υπάρχον όγκο δεδομένων. Παραδείγματα απο εξόρυξη γνώσης μπορεί να είναι η κατηγοριοποίηση κειμένων, η ομαδοποίησή τους καθώς και η εξαγωγή και ανάλυση συναισθήματος. Στην παρακάτω εικόνα παρουσιάζονται σχηματικά οι παραπάνω έννοιες:



Εικόνα 2.2.1-1: Κειμενική εξόρυξη γνώσης.

Οι πηγές κειμένου που βρίσκονται στην διαθεσή μας στο διαδίκτυο είτε από τα δωμάτια συζητήσεων (chat rooms) είτε απο τις ηλεκτρονικές βιβλιοθήκες καθώς και απο το ηλεκτρονικό μας ταχυδρομείο μπορούν να αποτελέσουν χρήσιμο εργαλείο σε πολλές εμπορικές εφαρμογές.’Ολος αυτός ο όγκος πληροφορίας που αποθηκεύεται ώς αδόμητο η ημιδομημένο κείμενο μπορεί να

βοηθήσει στην εξόρυξη πληροφορίας από κείμενα ο οποίος είναι ένας τομέας που αφορά την επιστήμη της ανάκτησης πληροφοριών, της μηχανικής μάθησης, της στατιστικής, της εξόρυξης δεδομένων, καθώς και της υπολογιστικής γλωσσολογίας.

Ο όρος ο οποίος χρησιμοποιείται περισσοτερο στην εργασία ειναι αυτός της Συναισθηματικής Ανάλυσης (Sentiment Analysis). Μια τεχνική στην εξόρυξη κειμένου είναι η εύρεση ομοιότητας των αρχείων κειμένου με κάποια θεματική περιοχή (στην κατηγοριοποίηση κειμένων) είτε και μεταξύ των αρχείων (στην ομαδοποίηση αρχείων κειμένου).
Καταλήγουμε οτι συνδυάζοντας τις δύο μεθόδους (Εξόρυξη Γνώμης & Ανάλυση Συναισθήματος) μπορούν να προσδιοριστούν τόσο το συναίσθημα είτε είναι θετικό είτε αρνητικό, η υποκειμενικότητα-αντικειμενικότητα αλλά και αλλου είδους συναισθήματα των χρηστών που δημιουργούν τις αναρτήσεις(θυμός, χαρά κλπ).

Όπως αναφέρθηκε και στην αρχή της εργασίας η εξόρυξη συναισθήματος βρίσκει εφαρμογή σέ πολλούς τομείς όπως τον πολιτικό, τον κοινωνικό, τον οικονομικό,τον καλλιτεχνικό και όλους τουσ τομείς που κατα καποιο τρόπο εξαρτώνται απο την κοινή γνώμη.

2.1 Άναλυση ορισμών και βασικής ορολογίας

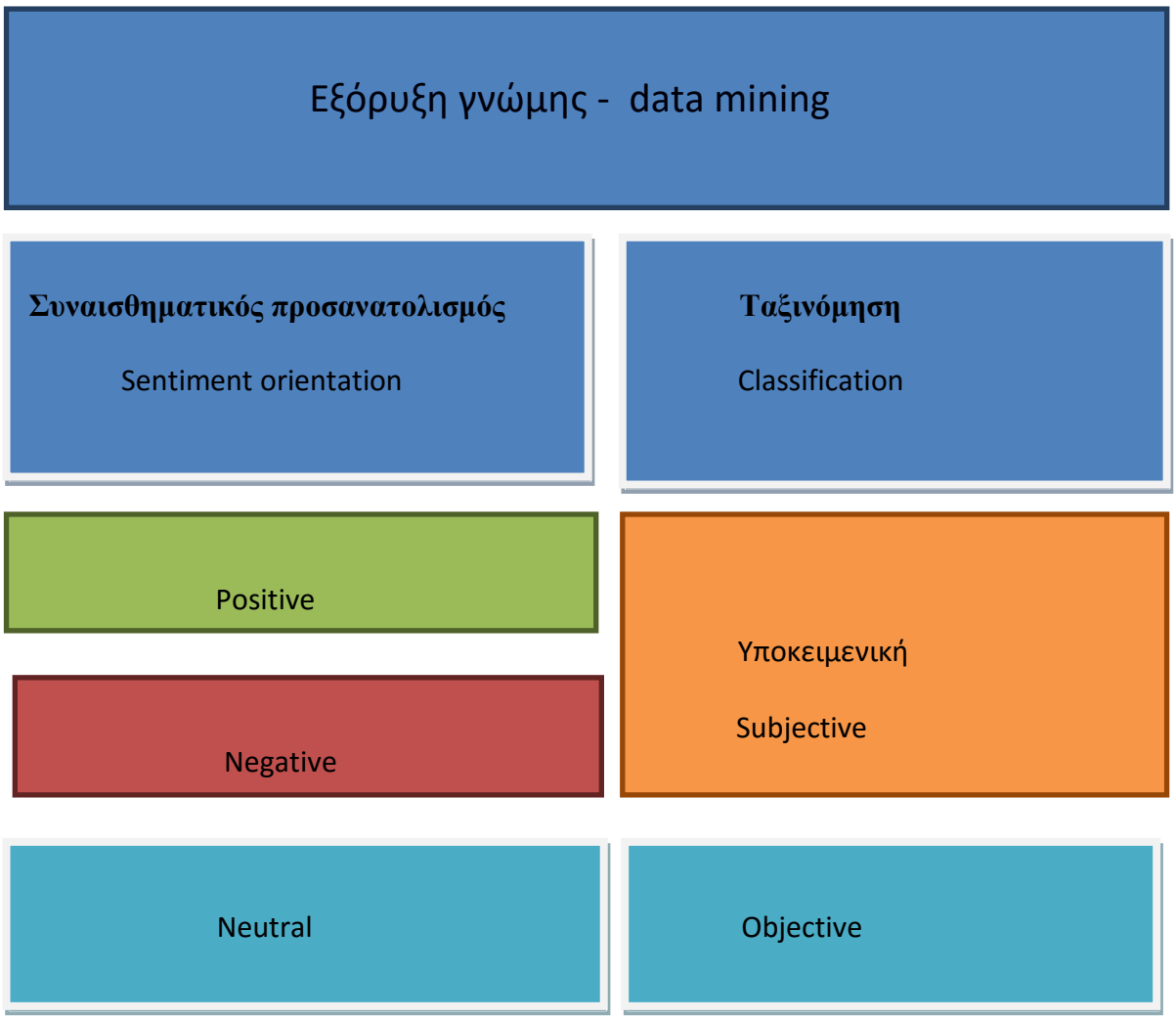
Το κεφάλαιο αυτό αφιερώνεται στην ανάλυση και ερμηνία των βασικότερων όρων που θα χρησιμοποιηθούν, για να γίνουν πιο κατανοητοί στον αναγνώστη.

Η σημασιολογική ανάλυση (text mining) εκτός των άλλων στοχεύει και στον προσδιορισμό μις άποψης που εκφράζεται σε ένα κείμενο, το οποίο περικλείει και την διαδικασία της διάκριση μεταξύ υποκειμενικής και αντικειμενικής άποψης [30].

Όταν αναφερόμαστε στην κατηγοριοποίηση κειμένου βάση της υποκειμενικότητας της, εννοούμε

την ταξινόμιση των προτάσεων σε δύο κλάσεις, υποκειμενικές ή αντικειμενικές. Αντικειμενικές θεωρούναι οι αμερόληπτες προτάσεις , ενώ υποκειμενικές όσες μεταφέρουν πληροφορίες με προσωπικές απόψεις. *Η τεχνική αναγνώρισης και ταξινόμησης των προτάσεων με άποψη ονομάζεται «Υποκειμενική Ταξινόμηση» (Subjectivity Classification)*

Υπάρχουν δυο είδη κατηγοριοποιήσης, η μία είναι σε όλο το έγγραφο (Document-level), και η άλλη γίνεται σέ κάθε πρόταση ξεχωριστά (Sentence level) κατηγοριοποιώντας μία μία τις προτάσεις σαν μία υποκειμενικές ή αντικειμενικές, και στην συνεχεια οι υποκειμενικές κατηγοριοποιούνται σε θετικές ή αρνητικές. Για να συμβεί αυτό, εφαρμόζονται τεχνικές επιβλεπόμενης μάθησης όπως ακριβώς και στην ταξινόμηση συναισθήματος εγγράφων καθώς και τεχνικές οι οποίες βασίζονται σε λεξικά συναισθήματος. Στη βιβλιογραφία συναντούμε συχνά τους συνώνυμους όρους «Συναισθηματική Κατηγοριοποίηση» (Sentiment Classification) και «Εξαγωγή Γνώμης» (Opinion Extraction).



Εικόνα 2.2.1-1: Υποκειμενική και αντικειμενική ταξινόμιση

Ορισμός : *Αντικειμενική - Υποκειμενική πρόταση:*
Η αντικειμενική πρόταση μεταφέρει αυτούσια μια πληροφορία, ενώ μια υποκειμενική εκφράζει συναίσθημα, απόψεις η και σκέψεις.

Ορισμός : *Πρόταση με άποψη:* είναι η πρόταση που εκφράζει άμεση ή έμμεση, θετική ή αρνητική άποψη. Μια τέτοια πρόταση μπορεί να κατηγοριοποιηθεί υποκειμενική είτε αντικειμενική.

ning) και την μέθοδο χρήσης λεξικών (lexicon) για να καταλήξουμε στη μέθοδο με τα καλύτερα αποτελέσματα για την διαδικτυακή εφαρμογή.

2.2.1 Μέθοδος χρήσης λεξικών

Η μέθοδος χρήσης λεξικών βασίζεται στη χρήση ειδικών λεξικών τα οποία περιέχουν προσημασμένες λέξεις, στις οποίες αποδίδονται βάρη ανάλογα με το νοήμα που έχουν και τί ένταση έχει η καθεμία στην χρήση της.
Η χρήση των λεξικών κατηγοριοποιείται ανάλογα με το άν

- Η χρήση λεξικών πραγματοποιείται ολιστικά, δηλαδή χρησιμοποιούνται στο σύνολο του κειμένου και υπολογίζεται με τη βοήθεια αλγορίθμου η θετική ή αρνητική στάση απέναντι στη θεματολογία (document level analysis),

- Ή η χρήση τους γίνεται τμηματικά, τα οποια διαχωρίζονται κατά την ανάλυση τους το οποίο

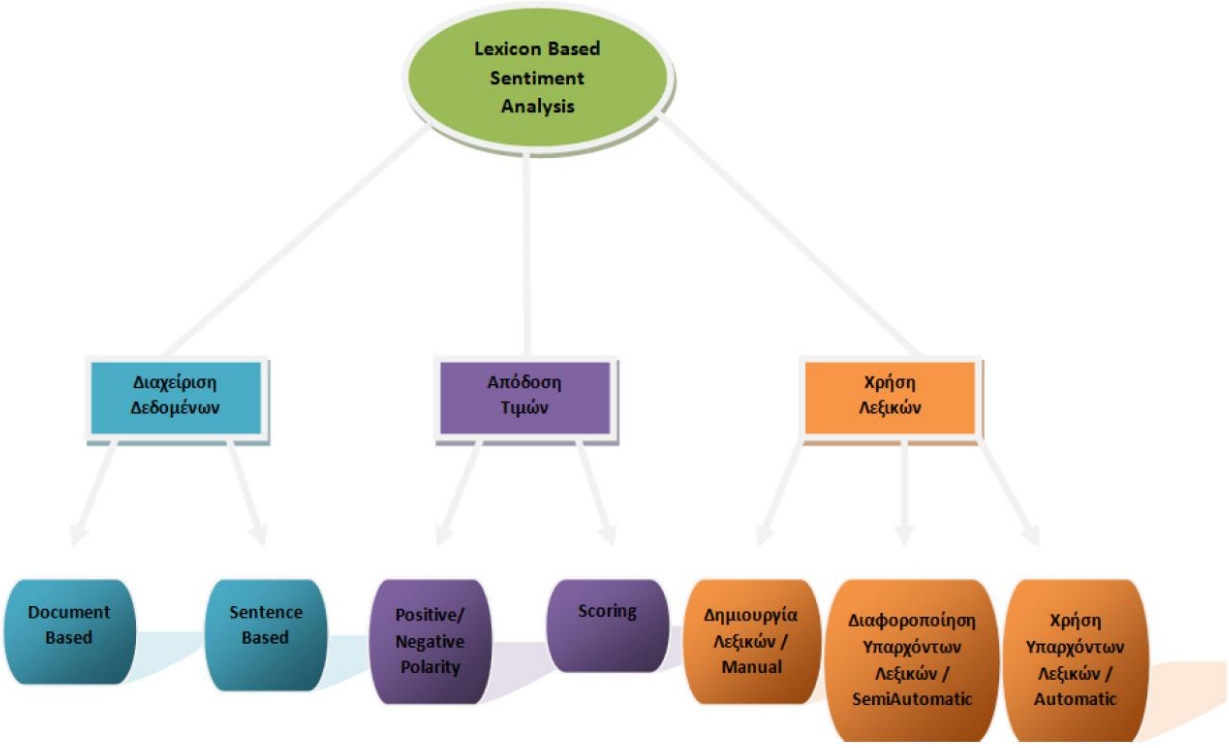
αποδίδει καλύτερα αν θέλουμε να εστιάσουμε σε συγκεκριμένη θεματολογία (sentence level analysis).

Η διαφορά στον τρόπο χειρισμού των λεξικών έχει να κάνει με τον τρόπο που μπορούμε να χειριστούμε τα δεδομένα. Μπορούμε να διαχωρίσουμε τα λεξικά και από τον τρόπο που το κάθε λεξικό χειρίζεται τα βάρη-τιμές που θα αποδωθούν στις λέξεις του και υπάρχουν οι εξείς δύο κατηγορίες:

1. Η πρώτη και η πιο απλή κατηγορία είναι αυτή της κατηγοριοποίησης των λέξεων σε αρνητικές και θετικές. Η λέξη “super” θεωρείται θετική, ενώ το “awful” είναι αρνητική. Έτσι οι λέξεις αποκτούν πολικότητα, και αποδίδεται σε όλο το κείμενο θετική ή αρνητική υποκειμενική άποψη.

2. Η δεύτερη μέθοδος αντί να ταξινομεί τις λέξεις τους αποδίδει τόσο αρνητικο όσο θετικό βάρος, ανάλογα με την σημασιολογία τους στις προτάσεις. Για παράδειγμα η λέξη “mad” είναι 0,4 positive και 0,6 negative, ενώ η λέξη “angry” είναι 1,0 negative και 0 positive. Ο αναμενόμενος χειρισμός σε αυτήν την περίπτωση μπορεί να πραγματοποιηθεί με δύο τουλάχιστον τρόπους:

Ακολουθεί ένα αντιπροσωπευτικό σχήμα οπου παρουσιάζονται οι κατηγοριοποιήσεις των λεξικών για τη σημασιολογική – συναισθηματική προσέγγιση.



Εικόνα 2.2.1-1: Η κατηγοριοποίηση λεξικών

2.2.2 Πλεονεκτήματα – Μειονεκτήματα χρήσης λεξικών

Το κυριότερο πλεονέκτημα της χρήσης των λεξικών είναι αρχικά η χαμηλή υπολογιστική ισχύς που χρειάζονται για να εξάγουν αποτελέσματα αλλά και το γεγονός οτι δεν χρειάζονται κανένα κείμενο για να εκπαιδευτούν, το οποίο γλυτώνει χρόνο και την ανθρώπινη εργασία. Η τεχνική αυτή χαρακτηρίζεται ως μη επιβλεπόμενη τεχνική μάθησης χρησιμοποιείται όταν δεν έχουν χτιστεί σύνολα εκπαίδευσης.

Για να εξάγει ένα λεξικό καλά αποτελέσματα πρέπει να επιλεχθούν με πολύ προσοχή οι λέξεις του. Πρέπει να προσεχθεί να μην χρησιμοποιηθούν λέξεις μόνο από συγκεκριμένη θεματολογία, διότι το λεξικό ειδικεύεται μόνο σε μία θεματολογία και θα χάνει τις υπόλοιπες. Για να είναι ένα λεξικό ολοκ-

ληρωμένο και να εξάγει αξιόλογα αποτελέσματα πρέπει να περιέχει λέξεις από πολύ μεγάλες συλλογές δεδομένων.

Ο συναισθηματικός προσδιορισμός που προέρχεται λεξικά εχει να κανει με πιθανοτητες (probabilistic results). Η σημασιολογία μιας πρότασης δεν εξαρτάται μόνο από την κατηγοριοποίηση των λέξεων ή/και των συνωνύμων τους σε θετικές ή αρνητικές. Η πρόταση «you are damn right» δεν μπορεί να ταξινομηθεί ως θετική, επειδή το επίθετο «damn» έχει κατηγοριοποιηθεί στο λεξικό ως αρνητική λέξη, η σημασιολογία των προτάσεων εξαρτάται και απο άλλους παράγοντες, όπως οι σύνδεσμοι και οι λέξεις. Οι τεχνικές και οι μέθοδοι ποικίλουν και ενδεχομένως, να πρέπει να εμπλουτιστούν ακόμα περισσότερο αλλά εν κατακλείδι είναι μια τεχνική που ενισχύει σημαντικά τις διαδικασίες της σημασιολογική ανάλυσης.

2.2.3 Μέθοδος Μηχανικής Μάθησης

Η μηχανική μάθηση (machine learning) ανήκει στο πεδίο της τεχνητής νοημοσύνης η οποία περιλαμβάνει δάφορους αλγορίθμους όπως και μεθόδους που δίνουν την δυνατότητα στους υπολογιστές να «μαθαίνουν». Η μηχανική μάθηση καθιστά εφικτή τη κατασκευή *προσαρμόσιμων* προγραμμάτων τα οποία λειτουργούν με βάση την αυτοματοποιημένη ανάλυση συνόλων δεδομένων και όχι τη διαίσθηση των μηχανικών που τα προγραμμάτισαν. Η μηχανική μάθηση μοιάζει σε μεγάλο βαθμό με τη στατιστική, αφού και τα δύο πεδία μελετούν την ανάλυση δεδομένων. Η χρήση των μεθόδων της μηχανικής μάθησης στην ανάλυση συναισθήματος και τη σημασιολογική ανάλυση κειμένων αποσκοπεί στον εντοπισμό και στην χρήση του κατάλληλου αλγόριθμου για την εξαγωγή αποτελεσμάτων.

Όπως είναι φυσικό όμως η εξαγωγή επιθυμητών αποτελεσμάτων απαιτεί πολλους πειραματισμούς

από τους ερευνητές του παιδίου με διαφορετικού τύπου αλγόριθμους, τους οποίους πρέπει να εκπαιδεύσουν σε πολλά και διαφορετικά σύνολα δεδομένων έτσι ώστε να κατηγοριοποιήσουν τις αγνώστες περιπτώσεις.

2.2.4 Είδη κατηγοριοποιήσεων κειμένου – Μέτρηση ακρίβειας

Ένα απο τα προβλήματα της εξόρυξης κειμένων είναι η εκτίμιση ομοιότητας μεταξύ εγγράφων διαφορετικού περιεχομένου. Αυτό σημαίνει είτε διαχωρισμό των εγγράφων σε προκαθορισμένες κατηγορίες είτε ομαδοποίηση εγγράφων σε φυσικές ομάδες.

Το προαπαιτούμενο βήμα για να καταλήξουμε στην επιλογή κατάληλου αλγορίθμου μηχανικής μάθησης, είναι η εξόρυξη κειμένου και η κατηγοριοποίηση του. Ένα σημαντικό προβλημα στην εξόρυξη κειμένου είναι να βρεθεί η ομοιότητα μεταξύ διαφορετικών εγγράφων.

Η κατηγοριοποίηση κειμένων είναι η διαδικασία κατα την οποία τα ηλεκτρονικά έγγραφα που είναι γραμμένα σε φυσική γλώσσα, ανατίθονται σε μια ή περισσότερες κατηγορίες, ανάλογα με το περιεχόμενό τους. Κατατάσουμε την κατηγοριοποίηση κειμένων στίς εξής τρεις κατηγορίες:

- Κατηγοριοποίηση με επίβλεψη (supervised classification), όπου κάποιος εξωτερικός μηχανισμός όπως ο άνθρωπος συμμετέχει στην σωστή κατηγοριοποίηση.

- Κατηγοριοποίηση με ημι-επίβλεψη (semi-supervised classification), όπου τμήματα των κειμένων έχουν ανατεθεί με ετικέτες από εξωτερικό μηχανισμό.

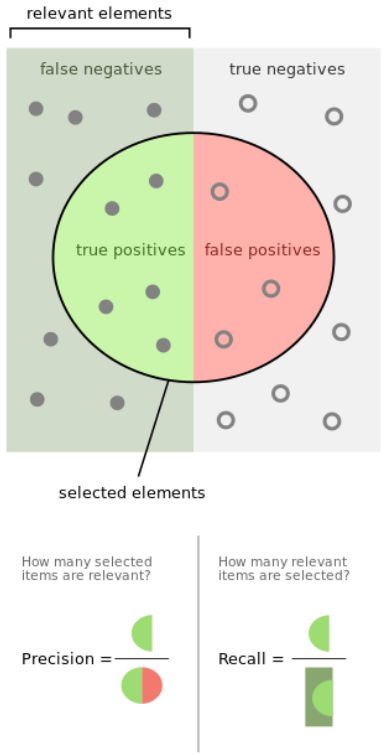
- Κατηγοριοποίηση χωρίς επίβλεψη (unsupervised classification), όπου η κατηγοριοποίηση γίνεται χωρίς καμία ανθρώπινη επίβλεψη.

Λόγω της αμφισιμότητας της γλώσσας και την δυσκολία να βρεθούν τα συναισήματα η μέθοδος κατηγοριοποίησης που χρησιμοποιήθηκε στην παρούσα εργασία είναι αυτή με επίβλεψη. Το σύνολο κειμένων για να γίνουν τα πειράματα χωρίζεται σε δύο τμήματα το στο σύνολο εκπαιδευσης (training set) και το σύνολο δοκιμής (test set).

Το training set πρέπει να είναι πάντα μεγάλο για αυτο αποτελεί συνήθως το 80% της συλλογής, ενώ το test set το έχουμε για να δοκιμάσουμε πόσο καλά λειτουργεί ο αλγόριθμος γία αυτό και αποτελέι το υπόλοιπο 20%. Το σύνολο δοκιμής ονομάζεται και ground truth διότι απο την πληροφορία που παρέχει εξαρτάται η ακρίβεια της κατηγοριοποίησης με επίβλεψη.

Τα έγγραφα του σύνολο εκπαίδευσης είναι γνωστό απο τον αλγόριθμο σε ποια κατηγορία ανήκουν αφου αυτά είναι τα δεδομένα με τα οποία λειτουργεί. Αρα είναι λογικό ότι όσο μεγαλύτερο είναι το σύνολο εκπαίδευσης τόσο πιο πολλά θα γνωρίζει ο αλγόριθμος και κατα συνέπεια θα αποδώσει καλύτερα. Ενώ για τα έγγραφα δοκιμής η πληροφορία της κατηγορίας υπάρχει αλλά δεν δίνεται στον αλγόριθμο. Ο αλγόριθμος αφού εκπαιδευτεί καλείται να τα κατατάξει στις σωστές κατηγορίες και στη συνέχεια τα αποτελέσματά του συγκρίνονται με τα πραγματικά δεδομένα.

Γνωστές τεχνικές κατηγοριοποίησης κειμένων είναι: κατηγοριοποιητής naïve Bayes, Latent Semantic Indexing (LSI), Support Vector Machines (SVM), tf-idf, τεχνητά νευρωνικά δίκτυα, αλγόριθμος k-κοντινότερων γειτόνων (kNN), concept mining και άλλοι. Οι κατηγοριοποιητές συνήθως δίνουν υψηλά ποσοστά ακρίβειας (high precision) αλλά χαμηλά ποσοστά ολοκλήρωσης (low recall). Οι όροι ακρίβεια (precision) και recall (ανάκληση) ορίζονται ως εξής:



Εικόνα 2.2.4-1: Ορισμός των precision και recall

Στην περίπτωση ενός ταξινομήτη που παίρνει την απόφαση για το αν ένα μήνυμα είναι θετικό ή αρνητικό ισχύουν τα εξής:

True positive: Το μήνυμα να ήταν θετικό και να προβλέφθηκε ως θετικό.

False positive: Το μήνυμα να ήταν θετικό και να μην προβλέφθηκε ως θετικό

False negative: Το μήνυμα να ήταν αρνητικό και να προβλέφθηκε ως θετικό

True negative: Το μήνυμα να ήταν αρνητικό και να προβλέφθηκε
ως αρνητικό

Οι δύο όροι ανάκληση και ακρίβεια είναι αντιστρόφως ανάλογοι, οπότε συνήθως υπολογίζεται η ακρίβεια σε διάφορα επίπεδα ανάκλησης. Το μέτρο F είναι ενδεικτικό της ακρίβειας του συστήματος, ως συνάρτησης των recall και Precision λαμβάνοντας τιμές από 0 έως 1 και υπολογίζεται ως εξής:

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Μια μέθοδος που έχει αξιόλογα αποτελέσματα σε ένα πεδίο μπορεί να μην είναι αξιόπιστη σε κάποιο άλλο. Το γεγονός αυτό οφείλεται στις ιδιαιτερότητες της φυσικής γλώσσας όπου η ερμηνεία πολλών λέξεων είναι αμφίσιμη.

3 *ΚΕΦΑΛΑΙΟ :* Αποτελέσματα από την Πειραματική Διαδικασία

Οι τεχνικές λεξικών που χρησιμοποιήσαμε έδειξαν ότι η χρήση ενός σημασιολογικά βαθμολογημένου λεξικού ενισχύει σημαντικά τις μεθόδους της υποκειμενικής ταξινόμησης. Αν και το λεξικό που

χρησιμοποιήσαμε επιδέχεται βελτιώσεις, παρόλα αυτά συνέβαλε σημαντικά στο υψηλό ποσοστό των κειμένων που ταξινομήθηκαν και επιπλέον δεν εξαρτάται από ένα σημασιολογικό πεδίο γεγονός που ενισχύει την χρηστικότητά του.

Όπως ανφέραμε και σε προηγούμενο κεφάλαιο οι τεχνικές βασισμένες σε λεξικά τό μόνο που απαιτούν είναι σύνολα δοκιμών και όχι εκπαίδευσης. Το σύνολο δοκιμής που θα χρησιμοποιηθεί σε αυτά τα πειράματα είναι 5 csv αρχεία με 30 αρχεία το καθένα.
Η συνέχεια του κεφαλαίου αποτελείται από την πειραματική διαδικασία που ακολουθήθηκε για να εξεταστούν οι δύο βιβλιοθήκες που επιλέξαμε να δοκιμάσουμε.

3.1 Ο αλγόριθμος "Sentimental" βασισμένος σε λεξικό

Πρόκειται για μια απλή βιβλιοθήκη που κάνει συναισθηματική ανάλυση, είναι γραμμένη εξ' ολοκλήρου στη γλώσσα προγραμματισμού Ruby. Μπορεί να εφαρμοστεί είτε σε επίπεδο πρότασης είτε σε επίπεδο κειμένου. Γίνεται διάσπαση στις προτάσεις και υπολογίζει την θετική ή την αρνητική πολικότητα, χαρακτηρίζει καθε πρόταση ξεχωριστά και στην συνέχεια ολόκληρο το κείμενο.

Η προεπιλεγμένη πολικότητα είναι το 0.0. Αν μια πρόταση βγάλει άθροισμα 0.0 τότε θεωρείται ότι δέν έχει συναίσθημα και χαρακτηρίζεται ως ουδέτερη. Μεγαλύτερο άθροισμα από το 0.0 θεωρεί θετικό το αποτέλεσμα ενώ μικρότερο αρνητικό. Υπάρχει η δυνατότητα αλλαγής της προεπιλεγμένης πολικότητας σε μη μηδενική τιμή όπως 0.5, άρα τα αποτελέσματα ορίζονται ως εξής:

- Θετικο άθροισμα > 0.5
- Ουδέτερα άθροισμα -0.5 - 0.5
- Αρνητικά άθροισμα < -0.5

3.1.1 Τα ενσωματωμένα συναισθηματικά λεξικά

Στην παρακάτω εικόνα εμφανίζονται πως είναι προσημασμένες οι λέξεις στην βιβλιοθήκη Sentimental. Συνολικά έχουν χρησιμοποιηθεί 18550 λέξεις όπου τους έχει αποδωθεί πολικότητα [-1, 1]

```
-1.0    )-:
-1.0    ):
-1.0    )o:
-1.0    8-0
-1.0    8/
-1.0    8\
-1.0    8c
-1.0    :'(
-1.0    :'-(
-1.0    :(
-1.0    :*(
-1.0    :,(
-1.0    :-(
-1.0    :-/
-1.0    :-S
-1.0    :-\
-0.50   :-|
-0.50   :/
-0.25   :0
-0.25   :S
-0.25   :\
-0.25   :|
-1.0    =(
-1.0    >:(
-1.0    D:
-1.0    sux
1.0     (o;
1.0     8-)
1.0     ;)
1.0     ;o)
1.0     %-)
1.0     (-:
1.0 :-)
1.0     (:
1.0     (o:
1.0     8)
1.0     :)
1.0     :-D
1.0     :-P
1.0     :D
1.0     :P
1.0     :P
1.0     :]
1.0     :o)
1.0     :p
1.0     ;^)
1.0     <3
1.0     &lt;3
1.0     =)
```

```
-0.416666666667 grumbling
-0.416666666667 fragile
-0.416666666667 endure
-0.416666666667 earthy
-0.416666666667 dilute
-0.416666666667 delinquent
-0.416666666667 defenseless
-0.416666666667 crude
-0.416666666667 crooked
-0.416666666667 bald
-0.416666666667 annoyance
-0.416666666667 angry
-0.416666666667 alien
-0.425  unresponsive
-0.425  unfriendly
-0.425  slack
-0.425  mindless
-0.4375 yahoo
-0.4375 wistful
-0.4375 whammy
-0.4375 wasteful
-0.4375 waive
-0.375  war
-0.4375 vulgar
-0.4375 vanity
-0.4375 unwell
-0.4375 untried
-0.4375 untested
-0.4375 unshaded
-0.4375 unquestioning
-0.4375 unpredictable
-0.4375 unofficially
-0.4375 uninvited
-0.4375 uninspired
-0.4375 unhappily
-0.4375 unfledged
-0.4375 undertow
-0.4375 underestimate
-0.4375 underage
-0.4375 uncovered
-0.4375 unconvincing
-0.4375 uncontrollable
-0.4375 unconditional
-0.4375 unclassified
-0.4375 uncharitable
-0.4375 unceremonious
-0.4375 uncalled-for
-0.4375 unawakened
-0.4375 unalterability
-0.4375 unaided
```

Εικόνα 3.1.1-1: Το προσημασμένο λεξικό του αλγορίθμου Sentimental

Εικόνα 3.1.1-2: Τα emoticons του αλγορίθμου Sentimental

Επειδή οι ιστοσελίδες και τα κοινωνικά δίκτυα όπως το twitter χρησιμοποιούν ειδικούς χαρακτήρες δηλαδή “φατσούλες” για να εκφράσουν συναίσθημα πολλά λεξικά χρησιμοποιούν τα ειδικά emoticons, η συγκεκριμένη χρησιμοποιεί 56 τέτοιους ειδικούς χαρακτήρες όπως φαίνεται και παραπάνω. Αυτή η ανάγκη προκύπτει από το γεγονός ότι στα κοινωνικά δίκτυα, που χρησιμοποιούν το Micro – Blogging, οι επιτρεπόμενοι χαρακτήρες ανάρτησης είναι περιορισμένοι. Έτσι λοιπόν, προκειμένου να γίνει πιο περιεκτική η επικοινωνία ανάμεσα στους χρήστες, αναπτύχθηκε και η χρήση των emoticons τα οποία μπορούν να εκφράζουν τα συναισθήματα των χρηστών με τη χρήση των ειδικών χαρακτήρων.

3.1.2 Αποτελέσματα πειραμάτων

Το πρώτο σύνολο πειραμάτων θα πραγματοποιηθεί με την Sentimental βιβλιοθήκη.
Η εξαγωγή του συναισθήματος θα γίνει μέσω της σύγκρισης των δεδομένων που έχουμε συλλέξει με τη χρήση του λεξικού που έχει η Sentimental. Όπως αναφέρθηκε και πιο πάνω το λεξικό έχει «Θετι-

κές» και «Αρνητικές» σημασιολογικά λέξεις. Υπάρχουν περιπτώσεις στις οποίες μπορεί να προκύψουν και μηδενικά σκόρ, αν πέσουμε στην περίπτωση να έχουμε τον ίδιο αριθμό θετικών και αρνητικών λέξεων.

Μετά τις προσθαφαιρέσεις προκύτουν οι τρείς παρακάτω κατηγορίες:

- *Θετική – Positivee*, αν το αποτέλεσμα της προσθαφαίρεσης είναι μεγαλύτερο από το 0.
- *Αρνητική – Negative*, αν το αποτέλεσμα της προσθαφαίρεσης είναι μικρότερο από το 0.
- *Ουδέτερη – Neutral*, αν το αποτέλεσμα της προσθαφαίρεσης είναι ίσο με το 0.

Παρακάτω παρουσιάζονται όλα τα πειράματα για όλα τα σύνολα δεδομένων, με τις εντολές που έτρεξαν και τους πίνακες με τα αποτελέσματα τους.

Το αρχείο "Gemfile" πού περιέχει τις βιβλιοθήκες που για να στήσουμε και να τρέξουμε το περιβάλλον του πρώτου συνόλου πειραμάτων με την βιβλιοθήκη Sentimental.

```
1  source 'http://rubygems.org'
2  gem 'sentimental'   //Η βιβλιοθήκη
3  gem 'pry'               //Εργαλείο για debugging
4  gem 'pry-rescue'
5  gem 'pry-stack_explorer'
```

Το εκτελέσιμο .rb που προγραμματίσαμε για να τρέξουμε τα πειράματα:

```
1  require 'csv'
2  require 'sentimental'
3  require 'open-uri'

4  csv_filename = ARGV[0]

5  def load_class_table(csv_filename)
6  csv_arxeio = File.new(csv_filename, "r")
7  # onoma arxeiou , class
8  table = CSV.read(csv_file, headers: true, header_converters: :symbol ,
   col_sep: " , ")
9  csv_file.close
10 return pinakas
11 end
12 def sentiment_classify(pinakas , classif)
13 pragmatika = 0
```

```
14 pinakas.each do |rep|
15 dokimi_object = open(rep[:filename]).read
16 dokimi_klasi = classif.get_senti(dokimi_object)
17 alithini_klasi = rep[:class].to_sym
18 if alithini_klasi == dokimi_klasi
      a. pragmatika += 1
19 end
20 end
21 akriveia = 0.0
22 akriveia = pragmatika / Float(pinakas.count)
23 return akriveia , pragmatika
24 end


25 def setup_classifier(dataset_filename = nil)
26 Sentimental.load_defaults
27 Sentimental.load_senti_file(dataset_filename) if dataset_filename
28 classif = Sentimental.new
29 return classif
30 end

31 dikimi_objects = load_class_table(csv_filename)

32 dokimi_objects.each do |rep|
33 puts "#{rep[:filename]} #{rep[:class]}"
34 end

35 classif = setup_classifier()
36 acc, tp = sentiment_classify(dokimi_objects,classif)
37 puts "Sunoliki akrivia: #{acc} kai #{tp} antikeimena katigoriopoihthikan
   swsta"
```

Για την διευκόλυνσή μας χρησιμοποιήσαμε διαφορετικό σύνολο βιβλιοθηκών (gems) που λέγεται gemspec, για να μην παρουσιαστεί κάποιο σφάλμα.

Το πρώτο πείραμα το παρουσιάζουμε σε printscreen για να φανεί ο τρόπος παρουσίασης των αποτελεσμάτων στο τερματικό. Έντεκα από τα είκοσι κατηγοριοποιήθηκαν σωστά με ποσοστό επιτυχίας 55%.

Εικόνα 3.1.2-1: Πρώτο πείραμα βιβλιοθήκης με το αρνητικό αρχείο

Συγκεντρωτικός Πίνακας Αποτελεσμάτων

Λόγω εξοικονόμισης χώρου αντί για την προβολή στιγμιότυπων οθόνης κάθε πειράματος ξεχωριστά επιλέξαμε να καταγράψουμε όλα τα αποτελέσματα της βιβλιοθήκης Sentimental σε έναν συγκεντρωτικό πίνακα που παρουσιάζονται στην συνέχεια.

Αλγόριθμος Sentimental		
Όνομα Αρχείου	Ποσοστό %	Σωστά ταξινομημένα αρχεία
Negative.csv	55 %	22/40
Negative_1.csv	60 %	24/40
Positive.csv	80 %	32/40
thetika_1.csv	75 %	30/40
Neutral.csv	0	0/40

Πίνακας 3.1.2-1: Αποτελέσματα του αλγορίθμου Sentimental

Όπως ήταν αναμενόμενο ο αλγόριθμος αυτος δεν κατηγοριοποιεί κανένα ουδέτερο κείμενο το οποίο οφείλεται στην πολικότητα, άν ανοίγαμε το εύρος της πολικότητας τα αποτελέσματα θα ήταν πιο ικανοποιητικά. Παρατηρούμε ότι τα θετικά κατηγοριοποιούνται καλύτερα από οτι τα αρνητικά.

3.2 *Ο αλγόριθμος "Sad Panda" βασισμένος σε λεξικό*

Αυτός είναι ο δεύτερος αλγόριθμος για συναισθηματική ανάλυση βασισμένος σε λεξικό που χρησιμοποιήσαμε, γραμμένος στη γλώσσα προγραμματισμού Ruby. Ο αλγόριθμος εκτός από την θετική και αρνητική ταξινόμιση κειμένου κάνει και κατηγοριοποίηση συναισθήματος, αναγνωρίζοντας τα εξής συναισθήματα: "Οργή", "αποστροφή", "χαρά", "έκπληξη", "φόβο" και "θλίψη". Οι λέξεις παίρνουν πολικότητα (polarity), με εύρος απο το 1 εως το 10 και εκφράζει την υποκειμενική άποψη, θετική ή αρνητική, που αποτυπώνεται στο σύνολο δεδομένων.

Το εκτελέσιμο πρόγραμμα που χρησιμοποιήσαμε για να τρέξουμε τα πειράματα:

```
1  require 'csv'
2  require 'sad_panda'

3  csv_filename = ARGV[0]

4  def load_class_table(csv_filename)
5  csv_file = File.new(csv_filename, "r")
6  # file name , class
7  table = CSV.read(csv_file, headers: true, header_converters: :symbol ,
   col_sep: " , ")
8  csv_file.close
9  return table
10 end
```

```
11 def sentiment_classify(table , classifier)
12 cor = 0
13 table.each do |tp|
14 test_object = open(tp[:filename]).read

15 #test_class = classifier.get_sentiment(test_object)
16 test_class = SadPanda.polarity(test_object)
17 real_class = tuple[:class].to_sym

18 acc = 0.0
19 acc = cor / Float(table.count)
20 return accuracy , correct
21 end
22 def setup_sentimental_classifier(dataset_filename = nil)
23 Sentimental.load_defaults
24 Sentimental.load_senti_file(dataset_filename) if dataset_filename
25 classifier = Sentimental.new
26 return classifier
27 end
28 if real_class == test_class
      a. correct += 1
29 end
30 t_objects = load_class_table(csv_filename)

31 t_objects.each do |tuple|
32 puts "#{tp[:filename]} #{tp[:class]}"
33 end

34 #classifier = setup_classifier()
35 #acc, tp = sentiment_classify(test_objects,classifier)
36 acc, tp = sentiment_classify(t_objects,nil)

37 puts "Overall accuracy: #{acc} and #{tp} objects were classified correctly"
```

3.2.1 Τα ενσωματωμένα συναισθηματικά λεξικά

Στη συνέχεια παρουσιάζεται η συναισθηματική προσέγγιση των λέξεων που χρησιμοποίηθηκαν στον συγκεκριμένο αλγόριθμο και η πολικότητα που τους δόθηκε. Ο αλγόριθμος ακόμα χρησιμοποιεί και μια μέθοδο που βρίσκει την υποκειμενικότητα δίνοντας βαρύτητα στις λέξεις (weaksubjectivity και strongsubjectivity), ανάλογα με το ποσο υποκειμενική είναι η λέξη. Τέλος μπορεί να αναγνωρίσει σε ένα κείμενο τα εξής συναισθήματα: "Οργή", "αποστροφή", "χαρά", "έκπληξη", "φόβο" και "θλίψη".

Κάθε λέξη που χρησιμοποιείται κατατάσεται σε ένα από τα παραπάνω συναισθήματα.

file | 1467 lines (1467 sloc) | 109.677 kb — Open | Edit | Raw | Blame | History | Delete

```
module TermPolarities
    # this method reads a csv file containing 'word,severity,positive/negative'
    # triplits, and returns a giant hash where the keys are individual words
        # and the values range between 0 and 2 (0 being more negative, 2 being most positive)
  def self.get_term_polarities
              @polarities = {"abandoned"=>2.5, "abandonment"=>2.5, "abandon"=>2.5, "abase"=>0,
                      "abasement"=>0, "abash"=>0, "abate"=>2.5, "abdicate"=>2.5, "aberration"=>0,
                      "abhor"=>0, "abhorred"=>0, "abhorrence"=>0, "abhorrent"=>0,
                      "abhorrently"=>0, "abhors"=>0, "abidance"=>10, "abide"=>10, "abject"=>0,
                      "abjectly"=>0, "abjure"=>2.5, "abilities"=>7.5, "ability"=>7.5, "able"=>7.5,
                      "abnormal"=>2.5, "abolish"=>2.5, "abominable"=>0, "abominably"=>0,
                      "abominate"=>0, "abomination"=>0, "above"=>7.5, "aboveaverage"=>7.5,
                      "abound"=>7.5, "abrade"=>2.5, "abrasive"=>0, "abrupt"=>2.5, "abscond"=>0,
                      "absence"=>2.5, "absentee"=>2.5, "absentminded"=>0, "absolve"=>10, "absurd"=>0,
                      "absurdity"=>0, "absurdly"=>0, "absurdness"=>0, "abundant"=>7.5,
                      "abundance"=>7.5, "abuse"=>0, "abuses"=>2.5, "abusive"=>0, "abysmal"=>0,
                      "abysmally"=>0, "abyss"=>0, "accede"=>10, "accept"=>7.5, "acceptance"=>7.5,
                      "acceptable"=>7.5, "accessible"=>7.5, "accidental"=>2.5, "acclaim"=>10,
                      "acclaimed"=>10, "acclamation"=>10, "accolade"=>10, "accolades"=>10,
                      "accommodative"=>7.5, "accomplish"=>7.5, "accomplishment"=>7.5,
```

Εικόνα 3.2.1-1: Απόδοση πολικότητας στις λέξεις απο 1-10

file | 6520 lines (6520 sloc) | 187.822 kb — Open | Edit | Raw | Blame | History | Delete

Search this file...

	abandoned	weaksubj	negative
2	abandonment	weaksubj	negative
3	abandon	weaksubj	negative
4	abase	strongsubj	negative
5	abasement	strongsubj	negative
6	abash	strongsubj	negative
7	abate	weaksubj	negative
8	abdicate	weaksubj	negative
9	aberration	strongsubj	negative
10	abhor	strongsubj	negative
11	abhorred	strongsubj	negative
12	abhorrence	strongsubj	negative
13	abhorrent	strongsubj	negative
14	abhorrently	strongsubj	negative
15	abhors	strongsubj	negative

Εικόνα 3.2.1-2: Λεξικό υποκειμενικότητας του αλγορίθμου

file | 1543 lines (1542 sloc) | 23.03 kb

Open | Edit | Raw | Blame | History | Delete

Search this file...

1	**abhor**	**anger**
2	abhorr	anger
3	abhorr	disgust
4	abhorrence	anger
5	abhorrent	disgust
6	abomin	anger
7	abomin	disgust
8	abominably	disgust
9	abominate	anger
10	abomination	anger
11	admir	joy
12	admir	surprise
13	admirable	joy
14	admirably	joy
15	admiration	joy
16	admiration	surprise

Εικόνα 3.2.1-3: Λεξικό συναισθημάτων του αλγορίθμου

3.2.2 Αποτελέσματα πειραμάτων

Παρακάτω παρουσιάζονται όλα τα πειράματα σε έναν συγκεντρωτικό πίνακα για όλα τα σύνολα δεδομένων.

Στην εικόνα 3.2.2-4 Βλέπουμε ένα πείραμα με το σύνολο κειμένων positive.csv το οποίο αποτελείται

απο είκοσι αρνητικά κείμενα και τα 18 από τα 20 κατηγοριοποιήθηκαν σωστά με ποσοστό επιτυχίας 90%

Εικόνα 3.2.2-1: Πείραμα του αλγορίθμου όπως αποτυπώνεται στην οθόνη

Πίνακας Αποτελεσμάτων

Αλγόριθμος Sad Panda		
Όνομα Αρχείου	Ποσοστό %	Σωστά ταξινομημένα αρχεία
Negative.csv	75 %	30/40
Negative_1.csv	60 %	24/40
Positive.csv	90 %	36/40
thetika_1.csv	70 %	28/40
Oudetera.csv	15.8%	6/40

Πίνακας 3.2.2-1: Τα αποτελέσματα του αλγορίθμου

Αυτός ο αλγόριθμος αποδίδει καλύτερα στην ταξινόμιση των θετικών και των αρνητικών και κατηγοριοποίησε ουδέτερα κείμενα αν και με πολύ χαμηλό ποσοστό, μολις 15.5%. Ο μόνος τρόπος για να έχουμε καλύτερη απόδοση στις παραπάνω βιβλιοθήκες είναι να αλλάξουμε τα λεξικά τους προσθέ-

τοντας άλλες πιο βαρυσήμαντες λέξεις που πιθανόν να έχουν περισσοτερο συναισθηματικό φορτίο. Επειδή όμως αυτός ο τρόπος είναι πολύ χρονοβόρος και δεν είναι σίγουρο οτι θα αποδώσει καλύτερα καταφεύγουμε στην επόμενη λύση που είναι ο κατηγοριοποιητής Naïve Bayes.

3.3 Κατηγοριοποιητής "Classifier"

Ο αλγόριθμος Classifier είναι προγραμματισμένος για Bayesian και άλλα είδη κατηγοριοποιήσεων. Ο αλγόριθμος Bayes (Naïve Bayes) είναι ένας κατηγοριοποιητής βασισμένος στις πιθανότητες. Η κατασκευή του εξαρτάται από ένα σύνολο εκπαίδευσης για να εκτιμήσει τις παραμέτρους μιας κατανομής πιθανότητας, δεδομένων των τιμών των χαρακτηριστικών ενός νέου εγγράφου. Αυτές οι πιθανότητες εκτιμώνται με τη βοήθεια του θεωρήματος Bayes.

Το εκτελέσιμο πρόγραμμα που χρησιμοποιήσαμε για να τρέξουμε τα πειράματα.

```
1  require 'csv'
2  require 'fast_stemmer'
3  require 'classifier'
4  csv_filename = ARGV[0]

5  def load_class_table(csv_filename)
6  csv_file = File.new(csv_filename, "r")
7  # file name , class
8  table = CSV.read(csv_file, headers: true, header_converters: :symbol ,
   col_sep: " , ")
9  csv_file.close
10 return table
11 end

12 def sentiment_classify(table , classifier)
13 cor_count = 0
14 table.each do |tp , assessor = classifier, count = cor_count|
```

```
acc = 0.0
acc= cor_count / Float(table.count)
return acc , cor_count
end

def setup_classifier(dtset = {})
thetika_files = load_class_table(dataset["positive"])
arnitika_files = load_class_table(dataset["negative"])

clas = Classifier::Bayes.new('negative','positive')

thetika_files.each do |tp|
train_object = open(tp[:filename]).read
   a. clas.train_positive(train_object)
end
neg_files.each do |tp|
train_object = open(tp[:filename]).read
   a. clas.train_negative(training_object)
end

return classifier
end

test_objects = load_class_table(csv_filename)

test_objects.each do |tuple|
puts "#{tp[:filename]} #{tp[:class]}"
end

clas = setup_classifier({"positive" => "positive.csv", "negative" => "nega-
tive.csv"})
acc, tp = sentiment_classify(test_objects,classifier) do |klasifier,
test_object , tp, cor_count|
puts klasifier
puts correct_count
test_class = klassifier.classify(test_object)
puts "#{real_class} #{test_class}"
correct_count
end

puts "Overall accuracy: #{acc} and #{tp} objects were classified correctly"
```

3.3.1 Data set - Συλλογές δεδομένων

Απο την θεωρία των Naïve Bayes είναι γνωστό οτι απαιτούν ένα ικανοποιητικό σύνολο εκπαίδευσης για να εξάγουν ικανοποιητικά αποτελέσματα. Από αυτή την θεωρία συνεπάγεται ότι η ποιότητα αλλά και η ποσότητα των συλλογών εκπαίδευσης (corpus) που χρησιμοποιούνται παίζουν πολύ σημαντικό ρόλο στην ακρίβεια των αποτελεσμάτων της μεθόδου. Ο εντοπισμός συναισθήματος σε ένα κείμενο απαιτεί την ανθρώπινη επίβλεψη λόγω των αμφίσημων νοημάτων της φυσικής γλώσσας για αυτό απαιτείται κατηγοριοποίηση με επίβλεψη για τον Naïve Bayes κατηγοριοποιητή. Για να πετύχουμε το σύνολο εκπαίδευσης που θα έχει τα πιο καλά αποτελέσματα πρέπει να χωρίζουμε το συνολο κειμένων σε δύο κατηγορίες, αυτό του συνόλου εκπαίδευσης και του συνόλου δοκιμής. Όποιο σύνολο εκπαίδευσης έχει τα καλύτερα αποτελέσματα θα επιλεχθεί και θα χρησιμοποιηθεί για τον κατηγοριοποιητή. Κατά την διάρκεια πειραμάτων παρατηρήθηκε ότι ο κατηγοριοποιητής εξάγει καλύτερα αποτελέσματα όταν τροφοδοτείται με πολλά είδη κειμένων. Έκτός απο τα κείμενα που συλλέχθηκαν χειροκίνητα από το διαδίκτυο χρησιμοποιήθηκε το polarity dataset v2.0 από το Movie Review Data [13] που αποτελείται απο 1000 θετικές και 1000 αρνητικές κριτικές ταινιών.

- thetika_web.csv (20 θετικά κείμενα τεχνολογικού περιεχομένου)
- arnitika_web.csv (20 αρνητικά κείμενα τεχνολογικού περιεχομένου)
- thetika_web_1.csv (60 θετικά κείμενα τεχνολογικού περιεχομένου)
- arnitika_web_1.csv (30 αρνητικά κείμενα τεχνολογικού περιεχομένου)
- thetika_last.csv (30 θετικά κείμενα τεχνολογικού περιεχομένου)
- cornell_thetika.csv (1000 θετικές κριτικές)
- cornell_arnitika.csv (1000 αρνητικές κριτικές)
- thetika_health.csv (30 θετικά κείμενα ιατρικού περιεχομένου)
- arnitika_health.csv (30 αρνητικά κείμενα ιατρικού περιεχομένου)

3.3.2 Πειράματα - Αποτελέσματα

Ακολουθεί όλη η πειραματική διδικασία με τα σύνολα εκπαίδευσης για τον κατηγοριοποιητή Naïve Bayes, για να καταλήξουμε στα πιο ικανοποιητικά αποτελέσματα.

Η λογική της εντολής για να «τρέξει» το πρόγραμμα:

rescue clas.rb TestSet.csv Positive TrainingSet.csv Negative TrainingSet.csv

Σύνολο Δοκιμής (Test set)	Σύνολο Εκπαίδευσης (Training Set)	
	Θετικά	**Αρνητικά**
thetika_last.csv	thetika_web_1.csv	arnitika_web_1.csv
	18 / 30	
arnitika_1.csv	thetika_web_1.csv	arnitika_web_1.csv
	49/60	
arnitika_1.csv	positive.csv	arnitika_web_1.csv
	0 / 60	
arnitika_1.csv	thetika_1.csv	arnitika_web_1.csv
	3 / 60	
arnitika_1.csv	thetika_web_1.csv	negative.csv
	60/60	
thetika_1.csv	positive.csv	arnitika_web_1.csv
	19 / 20	
thetika_1.csv	thetika_web_1.csv	arnitika_web_1.csv

11/20

Πίνακας 3.3.2-1: Πρώτο πείραμα

Σύνολο Δοκιμής (Test Set)	Σύνολο Εκπαίδευσης (Training Set)	
	Θετικά	**Αρνητικά**
arnitika.csv	thetika_web_1.csv	arnitika_web_1.csv
	58 / 60	
thetika.csv	thetika_web_1.csv	arnitika_web_1.csv
	8 / 60	

Πίνακας 3.3.2-2: Δεύτερο πείραμα με ίδιο training set

Σύνολο Δοκιμής (Test Set)	Σύνολο Εκπαίδευσης (Training Set)	
	Θετικά	**Αρνητικά**
arnitika_web_1.csv	thetika_web_1.csv	negative.csv
	3 / 30	
thetika_web_1.csv	positive.csv	arnitika_web_1.csv
	1 / 82	
arnitika.csv	positive.csv	arnitika_web_1.csv
	0 / 20	
thetika.csv	thetika_web_1.csv	negative.csv
	2 / 20	

Πίνακας 3.3.2-3: Τρίτο πείραμα

Όπως ήταν αναμενόμενο δέν ήταν όλα τα αποτελέσματα ικανοποιητικά, για αυτό έγινε αλλαγή στο σύνολο εκπαίδευσης για νέο σέτ πειραμάτων.

- all_positive.csv (thetika_1.csv+thetika_web_1.csv)
- all_negative.csv (arnitika_1.csv+ arnitika_web_1.csv)
- all_positive_1.csv (thetika.csv+ thetika_web_1.csv)
- all_negative_1.csv (arnitika.csv+ arnitika_web_1.csv)

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης	
	Θετικά	**Αρνητικά**
thetika.csv	all_positive.csv	all_negative.csv
	1 / 20	
thetika.csv	all_positive.csv	arnitika_web_1.csv
	10 / 20	
thetika.csv	all_positive.csv	Negative.csv
	18 / 20	

Πίνακας 3.3.2-4: Πρώτο πείραμα με θετικά σύνολα δοκιμής

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης	
	Θετικά	**Αρνητικά**
thetika_1.csv	all_positive_1.csv	all_negative_1.csv
	0 / 20	
thetika_1.csv	all_positive_1.csv	arnitika_web_1.csv
	5 / 20	
thetika_1.csv	positive	all_negative_1.csv
	19 / 20	

Πίνακας 3.3.2-5: Δεύτερο πείραμα με θετικα σύνολα δοκιμής

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης	
	Θετικά	**Αρνητικά**
arnitika_1.csv	all_positive_1.csv	all_negative_1.csv
	20 / 20	
arnitika_1.csv	all_positive_1.csv	arnitika_web_1.csv
	17 / 20	
arnitika_1.csv	positive.csv	all_negative_1.csv
	1 / 20	

Πίνακας 3.3.2-6: Πείραμα με αρνητικό συνόλο

Οι παρατηρήσεις που έχουμε να κάνουμε απο το πρώτο σετ πειραμάτων είναι οι εξής:

- Παρατηρούμε ότι τα θετικά εξαρτώνται από το σύνολο εκπαίδευσης τους και αποδίδουν πολυ καλά όταν έχουν κοινή θεματολογία (domain specific), δηλαδή έχουμε καλύτερα αποτελέσματα όταν το σύνολο εκπάιδευσης ανήκει στην ίδια κατηγορία με το σύνολο δοκιμής. Έχουμε χειρότερα αποτελέσματα όταν διαφέρει η κατηγορία του συνόλου εκπαίδευσης είναι απο αυτή του συνόλου δοκιμής.

- Τα αρνητικά σύνολα από την άλλη δέν έχουν το ίδιο πρόβλημα, ο αλγόριθμος δεν ειναι domain specific.

Για να δοκιμάσουμε το έτοιμο σύνολο κειμένων του Cornell [13] πρέπει να συνδυάσουμε όλα τα κείμενα που έχουμε συλλέξει. Ετσι δημιουργήθηκαν τα :

- finalpos.csv (περιέχει συνολικά 150 αρχεία)
- finalneg.csv (περιέχει συνολικά 90 αρχεία)

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης		Απόδοση %	Σύνολο
	Θετικά	**Αρνητικά**		
finalneg.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**63.3%**	**57/90**
finalpos.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**75,3%**	**113/150**
cornell_arnitika.csv	**finalpos.csv**	**finalneg.csv**	**97.2%**	**972/1000**
cornell_thetika.csv	**finalpos.csv**	**finalneg.csv**	**17.3%**	**173/1000**

Πίνακας 3.3.2-7: Πείραμα κειμένων Cornell ως training set

Από τους παραπάνω πίνακες παρατηρούμε ότι το σύνολο δοκιμής του Cornell αποδίδει λίγο καλύτερα στο δετικό σύνολο δοκιμής απο το αρνητικό. Ό μόνος τρόπος να καταλάβουμε γιατί έχει πιο εξασθενημένη απόδοση το σύνολο του Cornell είναι να διασπάσουμε τα σύνολα δοκιμών και να τα δοκιμάσουμε διαδοχικά.

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης		Απόδοση %	Σύνολο
	Θετικά	**Αρνητικά**		
thetika_web.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**96.6 %**	**29/30**
positive.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**90 %**	**18/20**
thetika_1.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**83.3 %**	**25/30**

thetika_health.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**70 %**	**21/30**
thetika_web_1.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**81.4%**	**57/70**

Πίνακας 3.3.2-8: Πρώτο πείραμα θετικών συνόλων δοκιμών

Χρησιμοποιώντας το σύνολο Cornell όλα τα υπόλοιπα σύνολα έχουν καλή απόδοση. Τα αρνητικά όπως και στο προηγούμενο σετ πειραμάτων αποδίδουν χειρότερα. Η θεωρία λέει οτί τα αρνητικά πάντα αποδίδουν χειρότερα από τα αρνητικά επειδή είναι πιο δύσκολο να "πιασει" ο αλγοριθμος τις αρνητικές έννοιες. Άλλοι λόγοι μπορεί να είναι η ανομοιότητα μεταξύ συνόλου δοκιμής και εκπαίδευσης στον η στον μικρό αριθμό του αρνητικού συνόλου εκπαίδευης.

Σύνολο Δοκιμής	**Σύνολο Εκπαίδευσης**		**Απόδοση %**	**Σύνολο**
	Θετικά	**Αρνητικά**		
arnitika_web.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**25 %**	**5/20**
negative**.csv**	**cornell_thetika.csv**	**cornell_arnitika.csv**	**85 %**	**17/20**
arnitika_1.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**100 %**	**20/20**
arnitika_health.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**36.6 %**	**11 / 30**
arnitika_web_1.csv	**cornell_thetika.csv**	**cornell_arnitika.csv**	**13%**	**4 /30**

Πίνακας 3.3.2-9: Δεύτερο πείραμα αρνητικών συνόλων δοκιμών

Με τις αλλαγές που έγιναν τα ποσοστά επιτυχίας στα αρνητικά αυξήθηκαν ενώ στα θετικά μειώθηκαν, με αυτά τα σύνολα εκπαίδευσης. Τα αποτελέσματα αυτά δεν μπορούν να είναι τα τελικά

που θα χρησιμοποιήσουμε αφού πρέπει να είναι πιο ισορροπημένα τα θετικά με τα αρνητικά.

Σύνολο Δοκιμής	Σύνολο Εκπαίδευσης	
	cornell_thetika	**finalneg_test**
arnitika_web.csv	**20/20**	
negative.csv	**3/20**	
arnitika_1.csv	**9/20**	
arnitika_web_1.csv	**29/30**	

Πίνακας 3.3.2-10: Πείραμα θετικού συνόλου με το νέο σύνολο εκπαίδευσης

Testset (Σύνολο Δοκιμής)	Trainingset (Σύνολο Εκπαίδευσης)	
	cornell_thetika	golneg_test
thetika_web.csv	0/20	
positive.csv	16/20	
thetika_1.csv	19/20	
thetika_web_1.csv	0/90	

Πίνακας 3.3.2-11: Δοκιμή αρνητικού συνόλου με νέο σύνολο εκπαίδευσης

Μετρώντας τα πειράματα με ποσοστο ακρίβειας μόνο δεν μπορούμε να κατανοήσουμε ακριβώς που δεν λειτουργεί καλά ο αλγόριθμός. Μετά απο καποιές αλλαγές στο πρόγραμμα η απόδοση του αλγορίθμου θα είναι ώς προς το precision και το recall, ορισμοί που αναλύσαμε και στο δεύτερο κεφάλαιο.

Ακολουθεί το τελικό σύνολο πειραμάτων με τις εντολές που χρησιμοποιήθηκαν καθώς και τα

αποτελέσματα σε πίνακες που εξήγαγε ο αλγόριθμος Naïve Bayes, τα οποία αποτυπώνονται ως προς την ακρίβεια (recall) και την ανάκληση (precision), καθώς επίσης μας δίνονται και τα λάθος θετικά (false positives) και λάθος αρνητικά (false negatives) κατηγοριοποιημένα κείμενα.

Εντολή: rescue test_oudetera.rb comb_test.csv comb_thetika.csv comb_arnitika.csv 0.04	
Overall clas. accuracy: 0.63 - 133/210	
ΘΕΤΙΚΆ	**ΑΡΝΗΤΙΚΑ**
Precision: 0.7	Precision: 0.6
Recall: 0.7	Recall: 0.9
false positives 21.0	false positives 27.0
false negatives 23.0	false negatives 4.0

Πίνακας 3.3.2-12: Πρώτο πείραμα με μετρήσεις ως πρός την ακρίβεια και την ανάκλησ

Εντολή: rescue test_oudetera.rb comb_test.csv comb_thetika_oudeteral.csv comb_arnitika.csv 0.04	
Overall clas. accuracy: 0.6 - 132/210	
ΘΕΤΙΚΆ	**ΑΡΝΗΤΙΚΑ**
Precision: 0.67	Precision: 0.6
Recall: 0.7	Recall: 0.85
false positives 27	false positives 23

false negatives 22	false negatives 7

Πίνακας 3.3.2-13: Δεύτερο πείραμα με μετρήσεις ως πρός την ακρίβεια και την ανάκληση

Εντολή: rescue test_oudetera.rb comb_test.csv comb_thetika.csv comb_arnitika_oudetera.csv 0.04

Overall clasaccuracy: 0.58- 125/210	
POSITIVE	**NEGATIVE**
Precision: 0.74	Precision: 0.5
Recall: 0.52	Recall: 0.95
false positives 14	false positives 46
false negatives 37	false negatives 2

Πίνακας 3.3.2-14: Τρίτο πείραμα με μετρήσεις ως πρός την ακρίβεια και την ανάκληση

Εντολή: rescue test_oudetera.rb comb_test.csv comb_thetika_oudetera.csv comb_arnitika_oudetera.csv 0.04	
Overall clas. accuracy: 0.60- 129/210	
ΘΕΤΙΚΆ	**ΑΡΝΗΤΙΚΑ**
Precision: 0.74	Precision: 0.5
Recall: 0.5	Recall: 0.95
false positives 14	false positives 46
false negatives 37	false negatives 2

Πίνακας 3.3.2-15: Τέταρτο πείραμα με μετρήσεις ως πρός την ακρίβεια και την ανάκληση

Εντολή: rescue test_oudetera.rb comb_test.csv comb_thetika_oudetera.csv comb_arnitika_oudetera.csv 0.03	
Overall clas accuracy: 0.60 - 129/210	
ΘΕΤΙΚΆ	**ΑΡΝΗΤΙΚΑ**
Precision: 0.7	Precision: 0.5
Recall: 0.6	Recall: 0.9

false positives 20	false positives 34
false negatives 29	false negatives 4

Πίνακας 3.3.2-16: Πέμπτο πείραμα με μετρήσεις ως πρός την ακρίβεια και την ανάκληση

3.4 Προβλήματα - Συμπεράσματα

Όπως ήταν φυσικό και γνωστό και από τις έρευνες άλλων επιστημόνων για να έχουν ικανοποιητικά αποτελέσματα οι κατηγοριοποιητές με λεξικά πρέπει να υπάρχουν τεράστιες συλλογές δεδομένων. Υπάρχουν όμως και τρόποι να κάνουμε και τα λεξικά να λειτουργήσουν καλύτερα χωρίς να τα εμπλουτίσουμε με νέες λέξεις. Ένας τρόπος είναι να αλλάξουμε την προεπιλεγμένη πολικότητα η οποία απλά μπορεί να βοηθήσει στην κατηγοριοποίηση και των μη συναισθηματικά φορτισμένων κειμένων (ουδέτερων). Παρατηρείται όμως και το φαινόμενο ακόμα και τα λεξικά που έχουν βασιστεί σε κατηγορία λέξεων ενός πεδίου να μην μπορούν να αποδώσουν εξίσου όταν χρησιμοποιούνται σε διαφορετικό πεδίο εφαρμογής. Επειδή όμως υποστηρίζεται ότι οι κατηγοριοποιητές λεξικών βασίζονται σε στατιστικούς τρόπους εξαγωγής αποτελεσμάτων προτιμήσαμε την χρήση του Naive Bayes κατηγοριοποιητή.

Το κυριότερο πρόβλημα που αντιμετωπίσαμε ήταν αυτό της συλλογής μεγάλου όγου δεδομένων, αφού η χρήση του Naive Bayes κατηγοριοποιητή απαιτεί κείμενα και για εκπαίδευση και για δοκιμή. Τα δεδομένα που συλλέξαμε προορίζονται το μεγαλύτερο μέρος τους για εκπαίδευση και το υπόλοιπο για δοκιμή. Το σύνολο δοκιμής που αποτελεί το 20-30% του συνόλου δεδομένων δεν αποτελεί πρόβλημα διότι ο αλγόριθμος πρέπει να εκπαιδευτεί με τον καταλληλότερο τρόπο για να επιτυγχάνει την καλύτερη κατηγοριοποίηση. Μία ιδιαιτερότητα που παρατηρήθηκε ήταν ότι κατά την εκπαίδευση του αλγορίθμου παρατηρήσαμε ότι το αρνητικό σύνολο εκπαίδευσης πρέπει να είναι μικρότερο απο αυτό του θετικού συνόλου για την επίτευξη του καλύτερου δυνατού αποτελέσματος. Μια ακόμη σημαντική παρατήρηση είναι ότι τα σύνολα εκπαίδευσης πρέπει να αποτελούνται από διάφορετικούς τομείς για να μην αποδίδει καλά ο κατηγοριοποιητής σε έναν συγκεκριμένο τομέα (domain specific). Σε γενικές γραμμές η κατηγοριοποίηση με επίβλεψη των κειμένων πρέπει να γίνεται προσεκτικά για να μην οδηγήσουμε τον αλγόριθμο σε εσφαλμένα αποτελέσματα.

Κατά την πειραματική διαδικασία παρατηρήσαμε ότι η ακρίβεια δεν ήταν αρκετό μέτρο σύγκρισης, για αυτό τροποποιήσαμε τον εκτελέσιμο κώδικα του αλγορίθμου έτσι ώστε μετράμε την απόδοση σε ακρίβεια (precision) και ανάκλήση (recall) καθώς και την εμφάνιση των λάθος θετικών και λάθος αρνητικών κατηγοριοποιημένων κειμένων, για να έχουμε καλύτερη γενική εικόνα των αποτελεσμάτων.

Οί λόγοι επιλογής του Naive bayes κατηγοριοποιητή ήταν όχι μόνο λόγω της δυνατότητας παραμετροποίησης του αλλά και της αξιοπιστίας των αποτελεσμάτων του.

Κατά την πειραματική διαδικασία είδαμε ότι η ακρίβεια δεν ήταν αρκετό μέτρο σύγκρισης, για αυτό τροποποιήσαμε τον κώδικα εκτέλεσης του αλγορίθμου έτσι ώστε να μας εξάγει τα αποτελέσματα σε ακρίβεια και ανάκλήση καθώς και την εμφάνιση των λάθος θετικών και λάθος αρνητικών κατηγοριοποιημένων κειμένων, για να έχουμε καλύτερη γενική εικόνα των αποτελεσμάτων.

Για τόν λόγο οτι μπορούμε να επέμβουμε στην πειραματική διαδικασία και τα αποτελέσματα της είναι πιο αξιόπιστα, ο αλγόριθμος ταξινόμησης που επιλέχθηκε για το χτίσιμο της εφαρμογής ήταν ο Naïve Bayes με σκοπό να υπολογιστεί η υπο συνθήκη πιθανότητα ένα μήνυμα d να είναι μέλος μίας κλάσης c, όταν οι πιθανές κλάσεις ήταν οι «αρνητικό» και «θετικό».

4 *ΚΕΦΑΛΑΙΟ :* Γενική Αρχιτεκτονική

4.1 Έρευνα Τεχνολογιών

4.1.1 Ruby [1]

Εικόνα 4.1.1-1: Λογότυπο της Ruby

Ο Yukihiro Matsumoto δημοσίευσε την πρώτη έκδοση της Ruby το 1995, χρησιμοποίησε κομμάτια από τις αγαπημένες του γλώσσες (Perl, Smalltalk, Eiffel, Ada και Lisp). Το 1999 αρχισε να γίνεται γνωστή επειδή δημιουργήθηκε το documentation στα αγγλικά. Είναι γλώσσα ανοιχτού κώδικα (open source) και είναι απλή στην κατανόηση και στη συγγραφή κώδικα. Ο βασικότερος λόγος της σημερινής της επιτυχίας στο διαδίκτυο, είναι το framework Ruby on Rails που ειναι γραμμένο σε Ruby. Συνεχίζει να αναπτύσεται λόγω της μεγάλης κοινότητας που έχει, και σήμερα βρίσκεται στην έκδοση 2.2.2. Ο δημιουργός της αναφέρει οτι η Ruby φτιάχτηκε για να κάνει ευτυχισμένους τους προγραμματιστές λόγω της απλότητας της. Κατά την συγγραφή κώδικα υπάρχουν λίγοι περιορισμοί και μπορούν να αναπτυχθούν μεγάλα και πολύπλοκα προγράμματα.

Το κυριότερο χαρακτηριστικό της είναι η εξ’ ολοκλήρου αντικειμενοστρέφεια της, τα πάντα στην Ruby θεωρούνται αντικείμενα ακόμα και οι αριθμοί και όλες της οι κλάσεις. Επειδή είναι επηρεασμένη από την Lisp έχει στοιχεία αναδρομικών γλωσσών (functional). Είναι πολύ ευέλικτη επειδή είναι διερμηνευτική (interpreted) γλώσσα. Ένα παράδειγμα της ευελιξίας της είναι οι ανοιχτές κλάσεις, δηλαδή η προσθήκη μεθόδων σε υπάρχουσες κλάσεις την ώρα εκτέλεσης ενός προγράμματος. Αυτή η δυνατότητα, θεωρείται από κάποιους αδυναμία γιατί ενδέχεται να γίνει μη ηθελημένη αντικατάσταση κάποιας μεθόδου. Ένα ακόμα από τα πιο γνωστά της στοιχεία, είναι η δυνατότητα «μετα-προγραμματισμού» ή αλλιώς η συγγραφή κώδικα που γράφει κώδικα.

4.1.2 Ruby On Rails [2]

Η Ruby on Rails δημιουργήθηκε το 2004, από την 37signals και τον David Heinemeier Hanson για την ανάπτυξη των εφαρμογών ιστού τους, είναι ένα από τα πρώτα frameworks ιστού τα οποία ακολούθησαν την αρχιτεκτονική Μοντέλο-Προβολή-Ελεγκτής MVC (Model-View-Controller).

Εικόνα 4.1.2-1: Λογότυπο RoR

Η RoR βασίστηκε σε ισχυρές αρχές και θεμέλια τα οποία ακολουθούνται πιστά από την κοινότητα η οποία βοηθάει στην ανάπτυξη της. Οι βασικότερες αρχές είναι:

- Ευέλικτες τεχνικές υλοποίησης (agile development)
- Σύμβαση αντί για παραμετροποίηση (convention over configuration)
- Ανάπτυξη εφαρμογών οδηγούμενες από δοκιμές (tests-driven development)

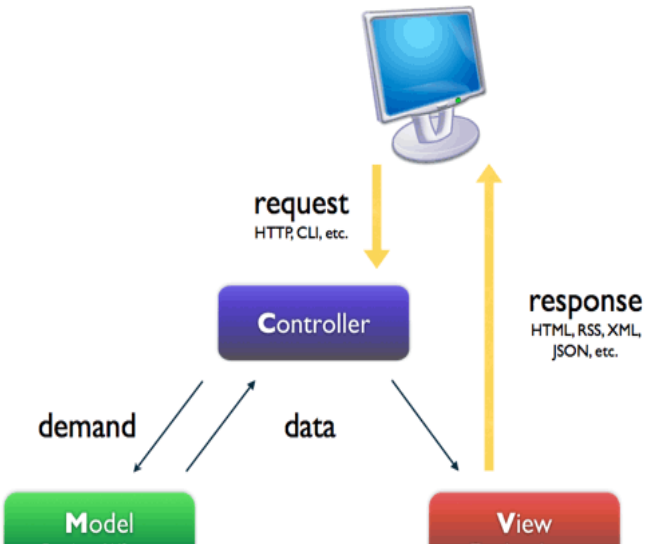
Στην Rails, το HTTP πρωτοκόλλο (GET, POST κλπ) σχετίζονται άμεσα με τα URLs και τις λειτουργίες

HTTP μέθοδοι	Διαδρομή	Λειτουργία βάσης	Τί κάνει
GET	/experiment	Read	Εμφανίζει μια λίστα πειραμάτων
GET	/ experiment /new	-	Επιστρέφει μια HTML φόρμα για τη δημιουργία νέου πειραματος
POST	/ experiment	Create	Δημιουργεί νέο πειραμα
GET	/ experiment /:id	Read	Εμφανίζεισυγκεκριμένο πειραμα
GET	/ experiment /:id/edit	-	Επιστρέφει μια HTML φόρμα για τη επεξεργασία συγκεκριμμένου πειραματος
PUT	/ experiment /:id	Update	Ενημερώνει τη βάση για τις αλλαγές στα δεδομένα συγκεκριμμένου πειραματος
DELETE	/ experiment /:id	Delete	Διαγράφει συγκεκριμένο πειραμα

Πίνακας 4.1.2-1: Αντιστοίχιση HTTP μεθόδων, Διαδρομών (paths), Λειτουργιών βάσης

4.1.2.1 Μοντέλο – Προβολή – Ελεγκτής

Πως εφαρμόζεται το μοντέλο (MVC) Μοντέλο – Προβολή – Ελεγκτής (Model – View – Controller) στο Ruby on Rails framework.



Εικόνα 4.1.2-2:Αναπαράσταση MVC αρχιτεκτονικής

Με αυτόν τον τρόπο η εφαρμογή διατηρείται καλά οργανωμένη χωρίζοντας τα δεδομένα (model) από τη λογική (controller) και την προβολή (view) του προγράμματος.

Τα μοντέλα περιέχουν τον κώδικα που ελέγχει και χειρίζεται τα δεδομένα ή ορίζει τη λογική. Τα μοντέλα είναι κλάσεις που επικοινωνούν με την βάση δεδομένων.

Οι ελεγκτές (controllers) απαντούν σε όλες τις αιτήσεις των χρηστών. Παίρνουν την είσοδο, την

διαχειρίζονται ανάλογα με το πώς έχουν προγραμματιστεί, καλούν μεθόδους, αλληλεπιδρούν με τα μοντέλα και προωθούν τα δεδομένα εξόδου στις προβολές (view). Οι προβολές εμφανίζουν τα δεδομένα εξόδου συνήθως σε HTML.

4.1.2.2 Δομή φακέλων της εφαρμογής RoR

Ένα απο τα πλεονεκτήματα των εφαρμογών σε Ruby on Rails είναι οτι από προεπιλογή με τη δημιουργία της είναι οργανωμένη σε φακέλους. Οι εικόνες, το σχήμα της βάσης δεδομένων, ο κώδικας HTML, CSS και κάθε άλλο διαφορετικό κομμάτι της εφαρμογής, βρίσκονται στην δική τους προκαθορισμένη θέση. Αυτή η λειτουργία συμβάλει στην εύκολη συντήρηση και επέκταση του κώδικα. Στην παρακάτω εικόνα παρουσιάζονται οι σημαντικότεροι φακελοι και αναλύονται οι λειτοθργίες τους.

app/ Ο φάκελος που χρησιμοποιείται περισσότερο από τους προγραμματιστές. Περιέχει τον κυρίως κώδικα της εφαρμογής (μοντέλο-προβολή-ελεγκτής).

config/ Οι επιπλέον ρυθμίσεις που απαιτούνται για την κάλυψη των αναγκών της εφαρμογής.

`config.ru` Αρχείο για την παραμετροποίηση της διεπαφής με τον Rack εξυπηρετητή.

db/ Σχήμα βάσης δεδομένων και πληροφορίες για τα migrations.

`Gemfile` Καθορίζονται τα gems τα οποία θα χρησιμοποιήσει η εφαρμογή. Τα gems είναι βιβλιοθήκες της Ruby.

`lib/` Εδώ υπαρχει κώδικας που είτε δεν ανήκει καπου ή χρησιμοποιείται από περισσότερα από ένα εκ των μοντέλων, προβολών, ελεγκτών (Model-View-Controller).

`log/` Όσο λειτουργεί μια Rails εφαρμογή δημιουργούνται καταγραφές (logs). Υπάρχουν τρεις φάκελοι για την ανάπτυξη, τις δοκιμές και την κατάσταση που η εφαρμογή έχει δημοσιευτεί.

`public/` Ο φάκελος αυτός είναι προσπελάσιμος από το διαδίκτυο και θεωρείται από τον εξυπηρετητή ο βασικός φάκελος της εφαρμογής και περιέχει στατικές σελίδες.

`Rakefile` Εδώ ορίζονται εργασίες που μπορεί να εκτελούν δοκιμές (tests), να δημιουργούν

τεκμηρίωση, να εκτυπώσουν το σχήμα της βάσης κ.α.

README Οδηγίες εγκατάστασης και χρήσης.

script/ Εδώ βρίσκονται τα scripts της RoR, τα οποία εκτελούνται αν στη κονσόλα γράψουμε την εντολή rails και το όνομα της συνάρτηση που θέλουμε να καλέσουμε π.χ *rails console*

test/ Σ' αυτό το φάκελο βρίσκονται όλες οι δοκιμές, όπως δοκιμές ενσωμάτωσης (integration tests), δοκιμές λειτουργικότητας (functional tests), δοκιμές επανάληψης (iteration tests).

tmp/ Ένας φάκελος για τα προσωρινά αρχεία, όπως τα περιεχόμενα της κρυφής μνήμης.

vendor/ Όλες σχεδόν οι εφαρμογές χρησιμοποιούν έτοιμο κώδικα ο οποίος προσθέτει λειτουργικότητα.

Εικόνα 4.1.2-3: Δομή φακέλων των εφαρμογών Ruby on Rails

4.1.2.3 Υποστήριξη τριών διαφορετικών περιβαλλόντων

Η Ruby on Rails υποστηρίζει τρια ανεξάρτητα περιβαλλόντα: ανάπτυξης, ελέγχων και παραγωγής. Οι ρυθμίσεις γίνονται μια φορά στην αρχική παραμετροποίηση και η μετάβαση από στάδιο σε στάδιο γίνεται με την αλλαγή μεταξύ περιβαλλόντων.

4.1.2.4 Active Record

Το Active Record είναι μια βιβλιοθήκη ORM της Ruby που επιτρέπει τη μεταφορά δεδομένων και συνεργάζεται με την υπόλοιπη εφαρμογή εξυπηρετώντας την καταχώρηση δεδομένων στη βάση, η οποία συνήθως είναι σχεσιακή. Το Active Record περιλαμβάνεται στη Ruby on Rails αλλά επίσης είναι διαθέσιμο και σαν Ruby Gem. Ακολουθώντας την αρχή των συμβάσεων αντί πραμετροποίησης ελαχιστοποιεί τις ρυθμίσεις που χρειάζεται να γίνουν, κάτι που δεν συναντάται συχνά στις υπόλοιπες ORM βιβλιοθήκες.

4.1.3 MySQL

Η βάση δεδομένων MySQL είναι μια απ' τις πιο δημοφιλής βάσεις δεδομένων. Είναι ανοιχτού κώδικα, σχεσιακή, αξιόπιστη, εύκολη στη χρήση, υψηλών επιδόσεων συμβατή με τις περισσότερες πλατφόρμες όπως τα Linux, Max OS, Windows κ.α.

4.1.3.1 Σχεσιακή βάση δεδομένων (Relational database)

Με τον όρο σχεσιακή βάση δεδομένων εννοείται μία συλλογή δεδομένων οργανωμένη σε συσχετισμένους πίνακες που παρέχει ταυτόχρονα ένα μηχανισμό για ανάγνωση, εγγραφή, τροποποίηση ή και πιο πολύπλοκες διαδικασίες πάνω στα δεδομένα. Ο σκοπός μιας βάσης δεδομένων είναι η οργανωμένη αποθήκευση πληροφορίας και η δυνατότητα εξαγωγής της πληροφορίας αυτής, ιδίως σε πιο οργανωμένη μορφή, σύμφωνα με ερωτήματα που τίθενται στη σχεσιακή βάση δεδομένων. Τα δεδομένα είναι δυνατόν να αναδιοργανώνονται με πολλούς διαφορετικούς τρόπους, σε νοητούς πίνακες, χωρίς να είναι απαραίτητη η αναδιοργάνωση των φυσικών πινάκων που τα αποθηκεύουν.

Επίσης, σε κάθε πίνακα υπάρχει μια στήλη το περιεχόμενο της οποίας είναι μοναδικό για κάθε γραμμή και ονομάζεται πρωτεύον κλειδί (primary key). Σύμφωνα με τις συμβάσεις της Rails, το πρωτεύων κλειδί είναι ακέραιος αριθμός και ονομάζεται id. Το σύνολο των πινάκων και η δομή τους αποθηκεύονται στο σχήμα της βάσης (database schema).

4.1.3.2 Αντικείμενο-σχεσιακή χαρτογράφηση (ORM – Object Relational Mapping)

Οι βιβλιοθήκες αντικειμενοσχεσιακής χαρτογράφησης είναι εργαλεία που επιτρέπουν την εύκολη και αυτοματοποιημένη αποθήκευση εγγραφών σε μια σχεσιακή βάση δεδομένων. Η λειτουργία τους είναι να κάνουν την αντιστοίχιση πινάκων με κλάσεις, γραμμών με αντικείμενα και στηλών με ορίσματα. Οι μέθοδοι των κλάσεων (class methods) επιδρούν σε πίνακες ενώ οι μέθοδοι των στιγμιοτύπων (instance methods) επιδρούν σε κάποια γραμμή του πίνακα.

Αποτέλεσμα της χρήσης ORM σε μια εφαρμογή είναι η μείωση του χρόνου ανάπτυξης του λογισμικού, του κόστους ανάπτυξης και συντήρησης, η σύνταξη απλούστερου και λιγότερου κώδικα αντί πολύπλοκων SQL ερωτημάτων.

4.1.4 Git

Εικόνα 4.1.3-1: Λογότυπο Git

Μια εφαρμογή κατά την ανάπτυξή της χωρίζεται από τον προγραμματιστή ή την ομάδα ανάπτυξης της σε στάδια – κατηγορίες. Το **Git** είναι ένα σύστημα ελέγχου διανεμόμενης έκδοσης και διαχείρησης κώδικα (SCM) με έμφαση στην ταχύτητα, στην ακεραιότητα των δεδομένων και στην υποστήριξη για κατανεμόμενες μη γραμμικές ροές εργασίας. Το Git σχεδιάστηκε και αναπτύχθηκε αρχικά από τον Λίνους Τορβαλντς για τη ανάπτυξη του πυρήνα του Λίνουξ το 2005 και έχει γίνει από τότε το πιο πλατιά διαδεδομένο σύστημα ελέγχου εκδόσεων για ανάπτυξη λογισμικού.

Όπως τα περισσότερα άλλα διανεμόμενα συστήματα ελέγχου εκδόσεων αναθεώρησης και αντίθετα με τα περισσότερα συστήματα πελάτη-διακομιστή κάθε κατάλογος εργασίας του Git είναι ένα ολοκληρωμένο αποθετήριο με πλήρες ιστορικό και δυνατότητες πλήρους παρακολούθησης της έκδοσης, ανεξάρτητα από την πρόσβαση δικτύου ή ενός κεντρικού διακομιστή.

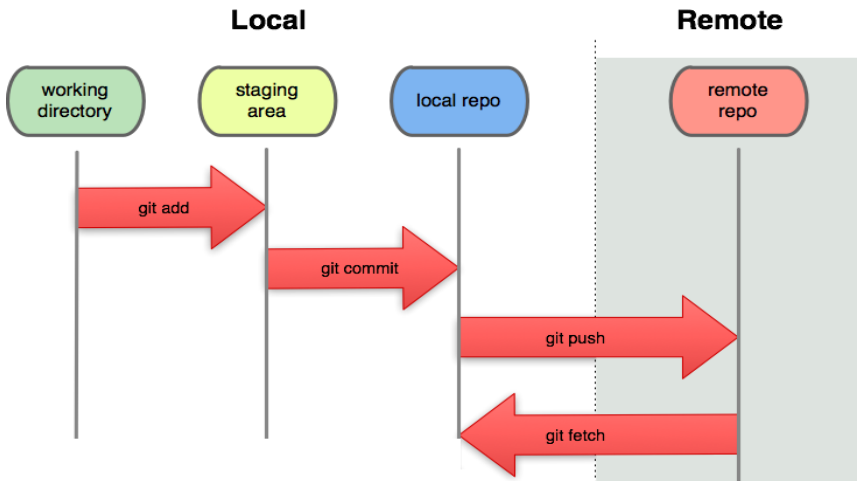
Οι προγραμματιστές να μπορούν να ανακαλέσει τις συγκεκριμένες εκδόσεις αργότερα. Από τη γέννησή του το 2005, το Git έχει εξελιχθεί, για να είναι εύκολο στη χρήση αλλά και να διατηρεί τις αρχικές του ιδιότητες. Είναι γρήγορο, πολύ αποτελεσματικό με τα μεγάλα έργα, και έχει ένα σύστημα διακλάδωσης για μη-γραμμική ανάπτυξη και χρησιμοποιείται από μεγάλες εταιρείες και μεγάλα έργα όπως οι Google, Microsoft, gnome, linux, twitter, facebook, LinkedIn, PostegreSQL, android, eclipse.

Η εγκατάσταση είναι εύκολη σε κάθε σύστημα εκτελώντας:

- `yum install git-core` (CentOS, fedora)
- `apt-get install git-core` (debian, ubuntu)
- `brew install git` (OS X)

Στα Windows, όπως ένα κοινό πρόγραμμα, μεταφορτώνεται και εγκαθίσταται το msysGit package.

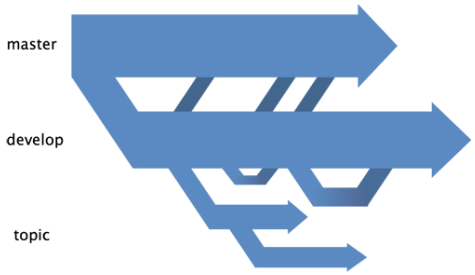
Τις περισσότερες φορές υπάρχουν ένα ή περισσότερα απομακρυσμένα αποθετήρια (remote repositories), εκτός από το τοπικό (local repository). Για να ενημερωθεί ένα απομακρυσμένο αποθετήριο για τις τελευταίες εκδόσεις εκτελείται η εντολή `git push`. Και αντίστροφα για την ενημέρωση του τοπικού αποθετηρίου από το απομακρυσμένο, εκτελείται η εντολή `git fetch`. Όμως με την εντολή `git fetch` ενημερώνεται το τοπικό αποθετήριο χωρίς να ενσωματώνει τις αλλαγές. Για την ενσωμάτωση η κατάλληλη εντολή είναι `git merge`.



Εικόνα 4.1.4-2: Ενημέρωση τοπικού και απομακρυσμένου repository

Όταν θέλουμε να πάρουμε ένα αντίγραφο τότε μπορούμε να κλονοποιήσουμε ένα απομακρυσμένο αποθετήριο μέ την εξής εντολή: `git clone` https://github.com

Το git, επίσης υποστηρίζει διακλαδώσεις (branches). Συνηθίζεται να υπάρχει μια κύρια διακλάδωση (master) η οποία περιλαμβάνει μόνο το κώδικα που προορίζεται για την τελική έκδοση της εφαρμογής κι άλλες διακλαδώσεις στις οποίες μπορεί να δοκιμάζεται κώδικας κι αν πληρεί τις προϋποθέσεις στη να προστεθεί στη κύρια διακλάδωση.



Εικόνα 4.1.4-3: Διακλαδώσεις στο Git

4.2 Επιλεγμένες Τεχνολογίες

4.2.1 Βάση δεδομένων – PostgreSQL

Η PostgreSQL αποτελεί μια ανοιχτού κώδικα σχεσιακή βάση δεδομένων. Η ανάπτυξη της ήδη διαρκεί πάνω από 20 χρόνια και βασίζεται σε μια αποδεδειγμένα καλή αρχιτεκτονική η οποία έχει δημιουργήσει μια ισχυρή αντίληψη των χρηστών της γύρω από την αξιοπιστία,την ακεραιότητα δεδομένων και την ορθή λειτουργία.

Εικόνα 4.2.1-1: Λογότυπο της PostgreSQL

Η PostgreSQL τρέχει σε όλα τα βασικά λειτουργικά συστήματα, περιλαμβάνοντας **Linux**, **UNIX** (AIX, BSD, Mac OS X, Solaris), και **Windows**.

Μερικές Γενικές οριακές Τιμές συμπεριλαμβάνονται στον παρακάτω πίνακα:

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32 TB
Maximum Row Size	1.6 TB
Maximum Field Size	1 GB
Maximum Rows per Table	Unlimited
Maximum Columns per Table	250 - 1600 depending on column types
Maximum Indexes per Table	Unlimited

Πίνακας 4.2.1-1: Οριακές τιμές ενεργών εγκαταστάσεων PostgreSQL

4.2.2 Twitter Bootstrap

Το Bootstrap είναι μια συλλογή εργαλείων ανοιχτού κώδικα (Ελεύθερο λογισμικό) για τη δημιουργία ιστοσελίδων και διαδικτυακών εφαρμογών. Περιέχει HTML και CSS για τις μορφές τυπογραφίας, κουμπιά πλοήγησης και άλλων στοιχείων του περιβάλλοντος, καθώς και προαιρετικές επεκτάσεις JavaScript. Έχει το πιο δημοφιλές πρόγραμμα στο GitHub και έχει χρησιμοποιηθεί από τη NASA και το MSNBC, μεταξύ άλλων.

Εικόνα 4.2.2-1: Λογότυπο Bootstrap

Το Bootstrap έχει σχετικά ελλιπή υποστήριξη για HTML5 και CSS, αλλά είναι συμβατό με όλους τους φυλλομετρητές (browsers). Βασικές πληροφορίες συμβατότητας των ιστοσελίδων ή εφαρμογές είναι διαθέσιμες για όλες τις συσκευές και τα προγράμματα περιήγησης.

Το Bootstrap είναι σπονδυλωτό και αποτελείται ουσιαστικά από μια σειρά στυλ(stylsheets) που εφαρμόζουν τα διάφορα συστατικά του πακέτου εργαλείων. Ένα στυλ που ονομάζεται bootstrap.less περιλαμβάνει τα συστατικά stylesheets. Οι προγραμματιστές μπορούν να προσαρμόσουν το αρχείο Bootstrap, επιλέγοντας τα στοιχεία που θέλουν να χρησιμοποιήσουν στο έργο τους.

Προσαρμογές είναι δυνατές σε περιορισμένη έκταση μέσω ενός κεντρικού στυλ διαμόρφωσης. Η χρήση γλώσσας στυλ επιτρέπει τη χρήση για μεταβλητές, λειτουργίες και φορείς (operators), ένθετους επιλογείς, γνωστά και ως μείγματα mixin.

Από την έκδοση 2.0, η διαμόρφωση του Bootstrap έχει επίσης μία ειδική επιλογή "Προσαρμογή " στην τεκμηρίωση (documentation). Επιπλέον, ο σχεδιαστής του έργου επιλέγει σε μια φόρμα τα επιθυμητά συστατικά και τα προσαρμόζει, εάν είναι αναγκαίο, σε τιμές διαφόρων εναλλακτικών λύσεων για τις ανάγκες του. Στη συνέχεια δημιουργείται ένα πακέτο που περιλαμβάνει ήδη το προχτισμένο CSS στυλ.

Εκτός από τα βασικά HTML στοιχεία, το Bootstrap περιέχει και άλλα στοιχεία περιβάλλοντος που χρησιμοποιούνται συχνά. Αυτά περιλαμβάνουν κουμπιά με προηγμένα χαρακτηριστικά (π.χ. ομαδοποίηση κουμπιών ή drop-down επιλογή, οριζόντιες και κάθετες καρτέλες, πλοήγηση, σελιδοποίηση, κ.λπ.), ετικέτες, προηγμένες τυπογραφικές δυνατότητες, εικονίδια, προειδοποιητικά μηνύματα και μια γραμμή προόδου.

4.2.3 Highcharts

Το Highcharts είναι μια βιβλιοθήκη διαγραμμάτων γραμμένη σε καθαρή JavaScript, προσφέροντας στον προγραμματιστή έναν εύκολο τρόπο να προσθέσει διδραστικά διαγράμματα στην ιστοσελίδα ή στην διαδικτυακή

Εικόνα 4.2.3-1: Λογότυπο

εγαρμογή του. Το Highcharts υποστηρίζει τους εξής τύπους
διαγραμμάτων : line, spline, area, areaspline, column, bar, pie,
Angular gauges polar chart types.
Οι χρήστες έχουν την δυνατότητα να εξάγουν το γράφημα απευθείας από την ιστοσελίδα. Είναι πλήρως συμβατή με όλους τους περιηγητές συμπεριλαμβάνοντας και τα iPhone/iPad και τον Internet Explorer από την έκδοση 6.

4.2.4 RVM [18]

Εικόνα 4.2.4-1: : Λογότυπο του RVM

Το RVM (Ruby Version Manager) δημιουργήθηκε από τον Seguin Waynee με σκοπό την ταυτόχρονη εγκατάσταση πολλών εκδόσεων της γλώσσας Ruby. Επίσης υποστηρίζει την καλύτερη οργάνωση των βιβλιοθηκών σε gemsets.Κάθε gemset αποτελείται από ένα σύνολο βιβλιοθηκών (gems) της Ruby. Ανάλογα με την εφαρμογή χρησιμοποιείται η κατάλληλη έκδοση βιβλιοθηκών και οργανώνονται καλύτερα οι αλληλεξαρτήσεις των βιβλιοθηκών.

4.2.5 Bundler

Ο Bundler αναπτύχθηκε από έναν από τους πρωτοπόρους της βασικής ομάδας της Rails και της jQuery, τον Yehuda Katz. Ελέγχει τις αλληλεξαρτήσεις μεταξύ των βιβλιοθηκών της Ruby και φροντίζει ώστε να γίνεται με αυτοματοποιημένο τρόπο η μεταφόρτωση όπως και η εγκατάσταση όλων των βιβλιοθηκών μιας εφαρμογής σε διαφορετικά συστήματα.

4.2.6 Λόγοι χρήσης επιμέρους τεχνολογιών

Ο κάθε προγραμματιστής επιλέγει τις τεχνολογίες και τα εργαλεία που κάνουν πιο εύκολη την δουλειά του. Σε αυτό το σημείο αναφέρουμε τους λόγους που κάνουν την Ruby on Rails ένα ισχυρό web framework γραμμένο σε Ruby.

Το γεγονός ότι το Ruby on Rails framework είναι καινούργιο όμως με τεράστια κοινότητα που το βοηθάει στην ανάπτυξη αποδείχτηκε εξαίρετη επιλογή καθώς έδωσε τη δυνατότητα στους προγραμματιστές να μάθουν τα βασικά χαρακτηριστικά σε πολύ μικρό χρονικό διάστημα και να μπορέσουν να δημιουργίσουν τα βασικά στοιχεία της εφαρμογής.

Κατά τη διάρκεια δημιουργίας της εφαρμογής αποδείχτηκε μεγάλη βοήθεια η τεράστια ποικιλία σε βιβλιοθήκες Gems που βοηθούν τον

προγραμματιστή στην ανάπτυξη σημαντικών χαρακτηριστικών του συστήματος.

Όσον αφορά τη βάση δεδομένων στα αρχικά στάδια, χρησιμοποιήθηκε η πιο διαδεδομένη σχεσιακή βάση δεδομένων, η MySQL. Όμως όταν έφτασε η στιγμή η εφαρμογή να περάσει στο στάδιο παραγωγής (development), η επιλογή της χρήσης του "υπολογιστικού νέφους" προέκυψε η ανάγκη για τη μετάβαση στη βάση δεδομένων PostgreSQL.
Αυτή η αλλαγή ήταν η καλύτερη επιλογή όσον αφορά το κομμάτι της αποθήκευσης των δεδομένων. Η εύκολη και ταχύτατη διαχείριση από τους προγραμματιστές, η σταθερότητα και αξιοπιστία μέσω του Heroku αλλά και η δυνατότητα επεκτασιμότητας είναι μερικοί από τους λόγους που κάνουν την PostgreSQL άριστη επιλογή πέρα από αναγκαία.

Λόγω του όγκου εργασίας που είχε η εφαρμογή απο μόνη της έπρεπε να βρεθεί μια ευκολη λύση

για τη δημιουργία ενός εύχρηστου και όμορφου περιβάλλοντος. Ύστερα από αρκετή μελέτη βρέθηκε η βιβλιοθήκη Twitter Bootstrap, η οποία μπορεί να βοηθήσει τον αρχάριο σχεδιαστή εφαρμογών διαδικτύου να αναπτύξει με λίγη προσπάθεια ένα καλαίσθητο αποτέλεσμα. Το Twitter Bootstrap προσφέρει μια συλλογή από εργαλεία που χρησιμοποιούν τεχνολογίες όπως HTML, CSS και JavaScript.

Ένα ακόμα χαρακτηριστικό της εφαρμογής είναι τα διαγράμματα. Μέσα από ένα διάγραμμα μπορεί ο χρήστης να κατανοήσει πώς απέδωσε ο αλγόριθμος σε σύγκριση με τις δικές του βαθμολογίες. Για την απεικόνιση των διαγραμμάτων χρησιμοποιήθηκε η βιβλιοθήκη Highcharts προσφέροντας γρήγορη απόδοση και όμορφο αποτέλεσμα. Η βιβλιοθήκη Highcharts παίρνει τις πληροφορίες που χρειάζεται από τη βάση δεδομένων.

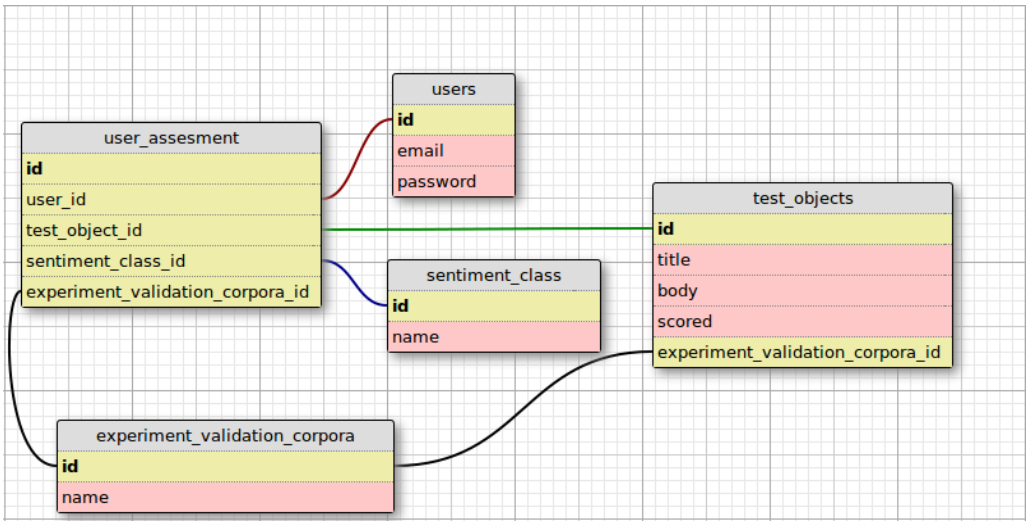
5 ΚΕΦΑΛΑΙΟ : **Η Εφαρμογή αξιολόγισης Analyment**

5.1 Η δημιουργία του σκελετού της εφαρμογής

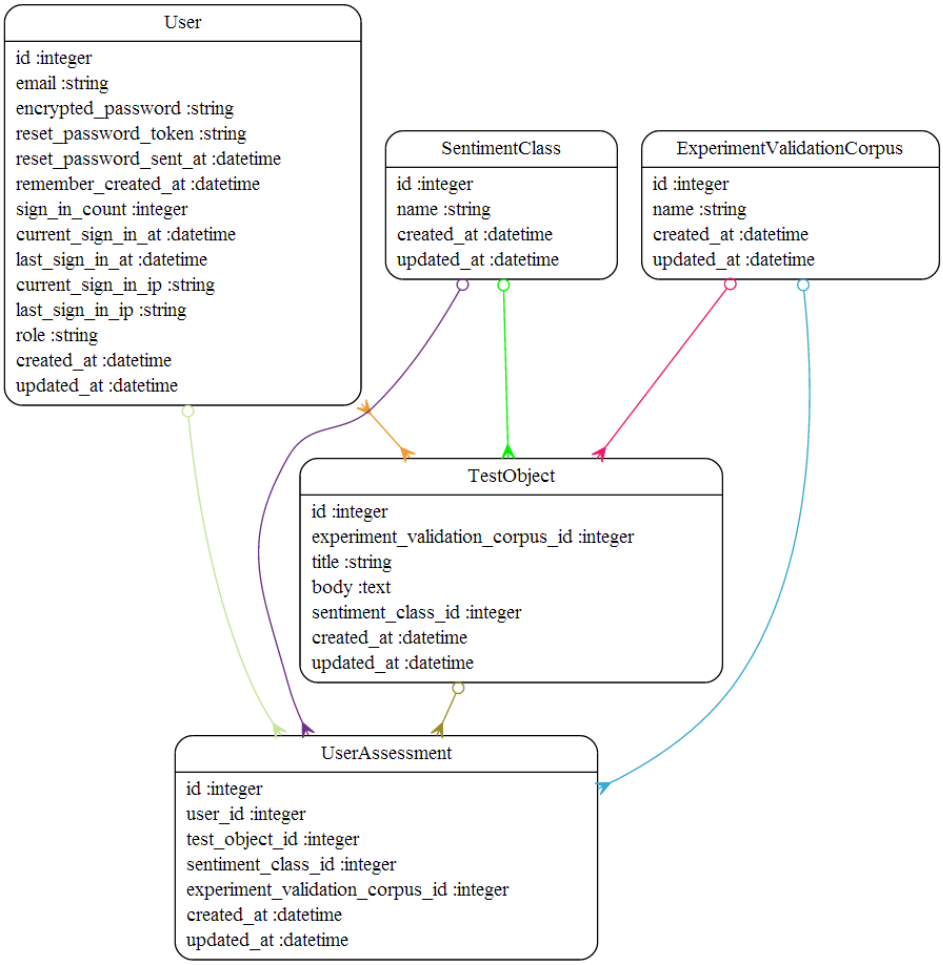
Η ιδέα για τη δημιουργία της εφαρμογής η οποία στοχεύει στην αξιολόγηση των αλγορίθμων που κάνουν συναισθηματική ανάλυση προήλθε από το γεγονός ότι ο ανθρώπινη νοϋμοσύνη είναι ακόμα σε πολύ υψηλότερα επίπεδα από αυτά των μηχανών. Έτσι ο άνθρωπος καλείται να βαθμολογήσει

κείμενα τα οποία έχουν βαθμολογιθεί απο έναν αλγόριθμο πρώτα.

Ο κορμός της εφαρμογής είναι η βάση δεδομένων της, δηλαδή οι πίνακες που περιέχει. Αρχικά δημιουργήθηκαν οι πίνακες: Πίνακας των κατηγοριών, της συναισθηματικής κλάσης, κειμένου δοκιμής , και ανάθεσης χρήστη. Ο χρήστης έχει τη δυνατότητα να επιλέγει την κατηγορία του πειράματος και να αξιολογήσει τα κείμενα της κατηγορίας αυτής.



Εικόνα 4.2.6-1: Το αρχικό Schema βάσης δεδομένων



Εικόνα 4.2.6-3: Schema βάσης δεδομένων προγραμματιστικά

Στο δεύτερο βήμα δημιουργήθηκαν οι αλγόριθμοι για να υπολογίζουν τις πληροφορίες που εισάγει ο χρήστης. Οι πληροφορίες αυτές είναι οι εξής:

- Το σύνολο κειμένων που αξιολόγισε ο χρήστης
- Σύγκριση των αποτελεσμάτων
- Τελικά αποτελέσματα με διαγράμματα

Τέλος δημιουργίθηκε το περιβάλλον διεπαφής των χρηστών. Όλα τα παραπάνω προεκτάθηκαν για πολλούς χρήστες μέσω του πίνακα χρηστών και της διαδικασίας διαπίστευσης χρηστών. Επίσης δεν δώθηκαν δικαιώματα χρήσης των παραπάνω στους μη συνδεδεμένους χρήστες.

5.2 Τα αρχεία που απαρτίζεται η εφαρμογή

Τα gems που χρησιμοποιήθηκαν είναι τα παρακάτω:

```
source 'https://rubygems.org'
# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.0.2'
# Use postgreSQL as the database
gem 'mysql2'
# Application server
gem 'puma'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 4.0.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
gem "therubyracer"
gem "less-rails" #Sprockets (what Rails 3.1 uses for its asset pipeline) supports LESS
gem "twitter-bootstrap-rails"
# Imports csv files
gem 'roo'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby
# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read
more: https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 1.2'
```

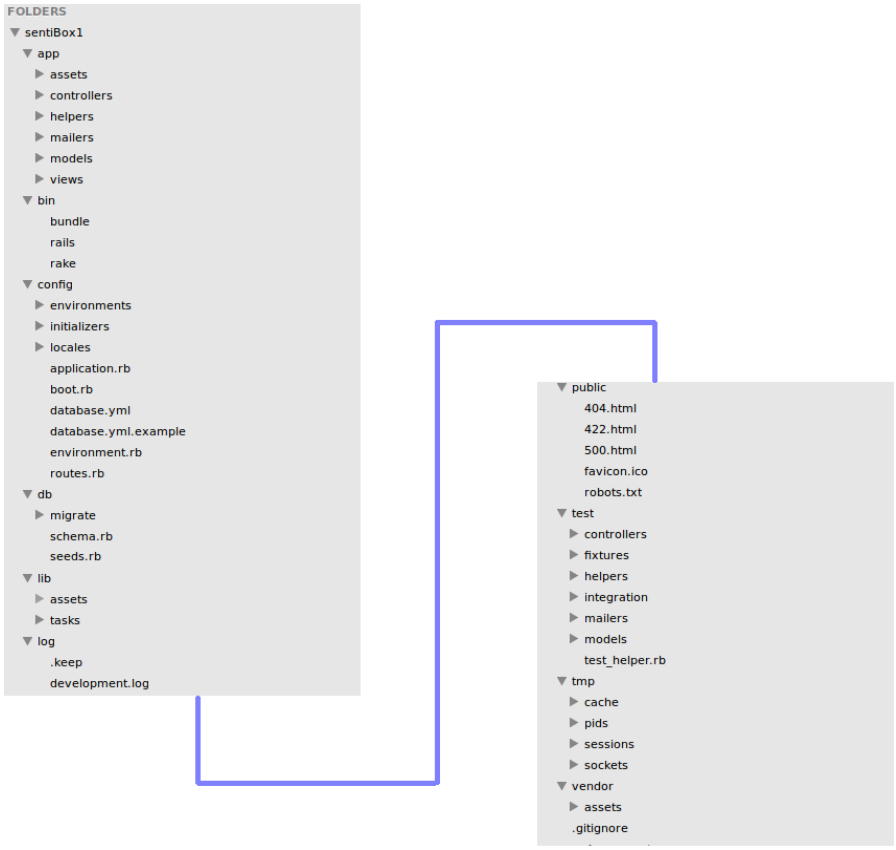
```
# User authentication
gem 'devise'
gem 'bcrypt-ruby', '~> 3.1.2'
#User authorization
gem 'cancan'
# Pagination
gem 'kaminari'
group :doc do
  # bundle exec rake doc:rails generates the API under doc/api.
  gem 'sdoc', require: false
end
group :development do
    # Debugging tool
    gem 'pry'
end
# Use ActiveModel has_secure_password
# gem 'bcrypt-ruby', '~> 3.1.2'

# Use unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano', group: :development

# Use debugger
# gem 'debugger', group: [:development, :test]
ruby "2.0.0"
```

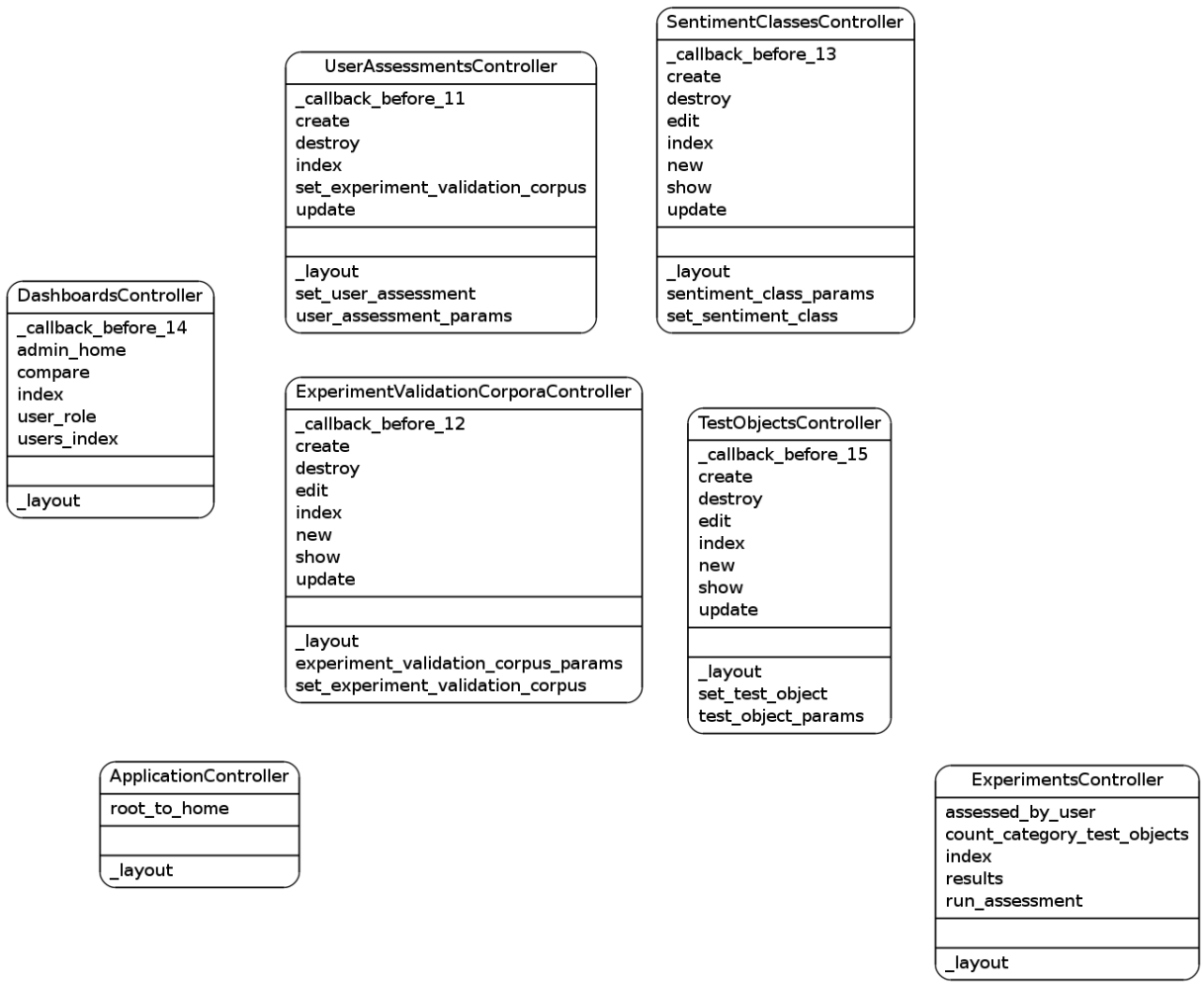
Στην παρακάτω εικόνα φαίνονται τα αρχεία της εφαρμογής:



Εικόνα 4.2.6-1: Δομή φακέλων της εφαρμογής Analyment

5.3 Πώς σχέδιάστηκε η εφαρμογή

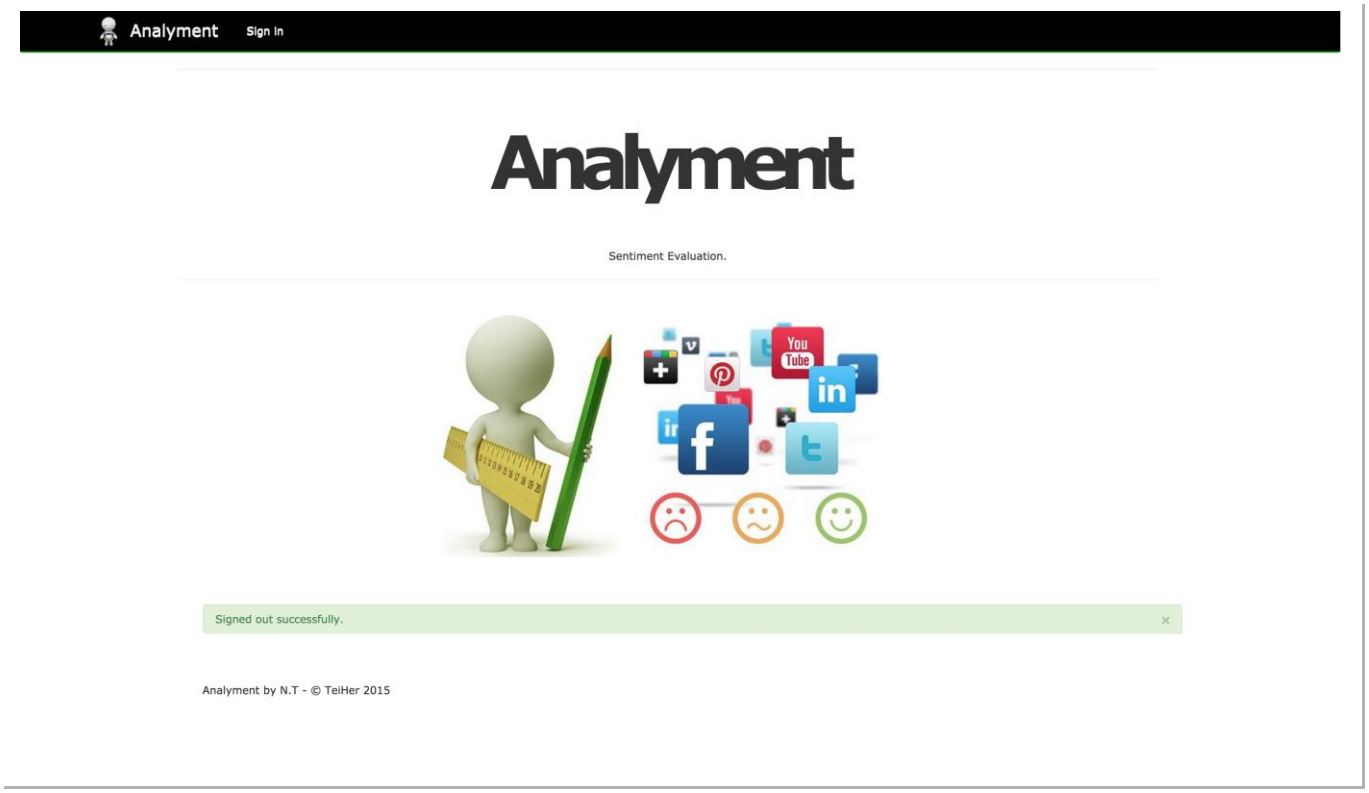
Αφού οι controller αποτελούν την το κέντρο της εφαρμογής μίας και συνδέουν όλες τις λειτουργίες όπως τη βάση με τους προβολείς παραθέτουμε την εικόνα με τον σχεδιασμό τους. Το διάγραμμα κλάσεων για τους controller δημιουργήθηκε χρησιμοποιώντας μόνο μία βιβλιοθήκη της Ruby η οποία μόλις την εκτελέσεις παράγει την εικόνα.



Εικόνα 4.2.6-1: Δομή και σύνδεση των controllers

5.4 Λειτουργίες χρηστών

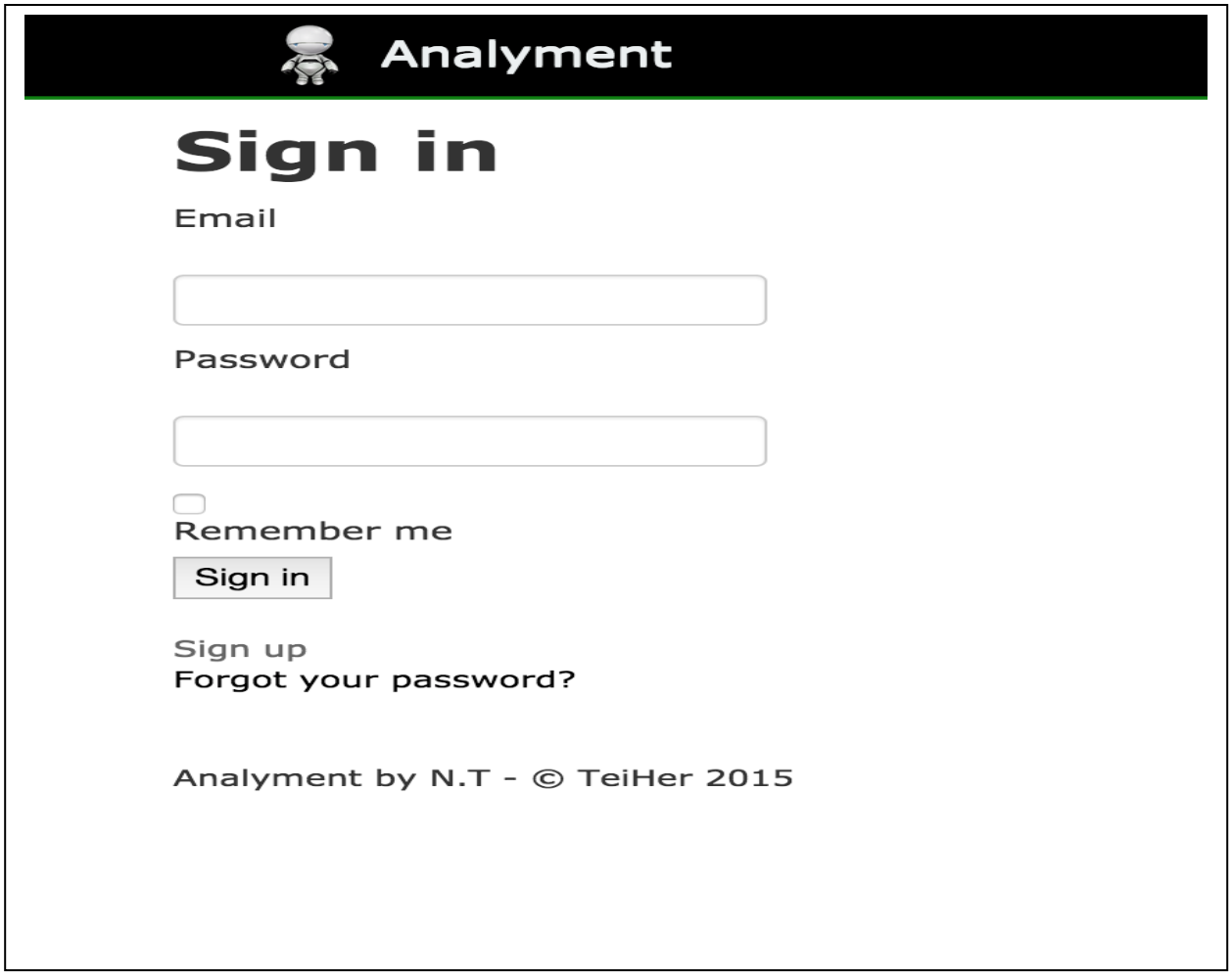
Η αρχική σελίδα που εμφανίζεται σε εγγεγραμένους και μη εγγεγραμμένους, και τους δίνει την δυνατότητα εγγραφής (sign up) καθώς και η δυνατότητα σύνδεσης (sign in) στο σύστημα από την μπάρα που βρίσκεται στην κορυφή της εφαρμογής.

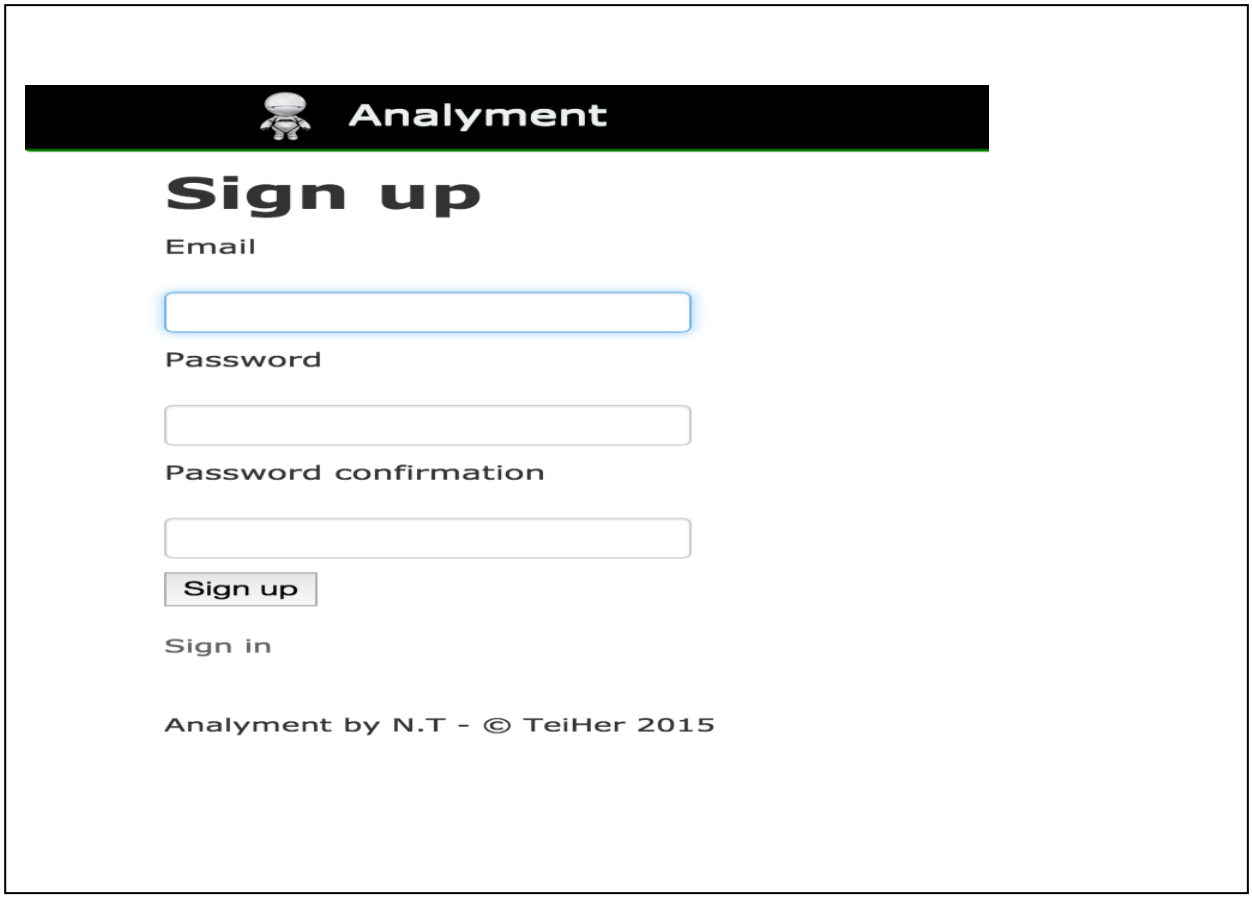


5.4.1 Δημιουργία – σύνδεση χρήστη

Ολόκληρη η εφαρμογή είναι σχεδιασμένη και φτιαγμένη να είναι όσο πιο εύκολη στην χρήση της. Ο απλός χρήστης μπορεί να ανοίξει πολύ εύκολα λογαριασμό στην εφαρμογή Analyment. Στην κορυφή της αρχικής σελίδας επιλέγοντας το κουμπί sign up εμφανίζεται η φόρμα συμπλήρωσης με τρία υποχρεωτικά πεδία:

- E-mail. Υποχρεωτικά ενα έγκυρο e-mail το οποίο θα χρησιμοποιείται για τη σύνδεση του χρήστη.
- Password (κωδικός). Ο μυστικός κωδικός τουλάχιστον 8 χαρακτήρων που θα χρησιμοποιηθεί σε συνδυασμό με το e-mail για τη σύνδεση του χρήστη στο σύστημα.
- Password Confirmation (επιβεβαίωση κωδικού). Για την επανάληψη του κωδικό του για να αποφευχθούν τυχόν λάθη.


Analyment
Sign in
Email
Password
Remember me
Sign in
Sign up
Forgot your password?
Analyment by N.T - © TeiHer 2015


Analyment
Sign up
Email
Password
Password confirmation
Sign up
Sign in
Analyment by N.T - © TeiHer 2015

Μόλις κάποιος χρήστης δημιουργήσει τον λογαριασμό του μπορεί να συνδεθεί στην εφαρμογή Analyment μόνο με το e-mail και τον κωδικό του.

Επιλέγοντας το Remember_me αποθηκεύεται στην σελίδα και δεν εισάγει κάθε φορά τα στοιχία του. Υπάρχει και η επιλογή του forgot your password όπου ο χρήστης έχει τη δυνατότητα να ανακτήσει τον κώδικό του εισάγοντας μόνο το email του.

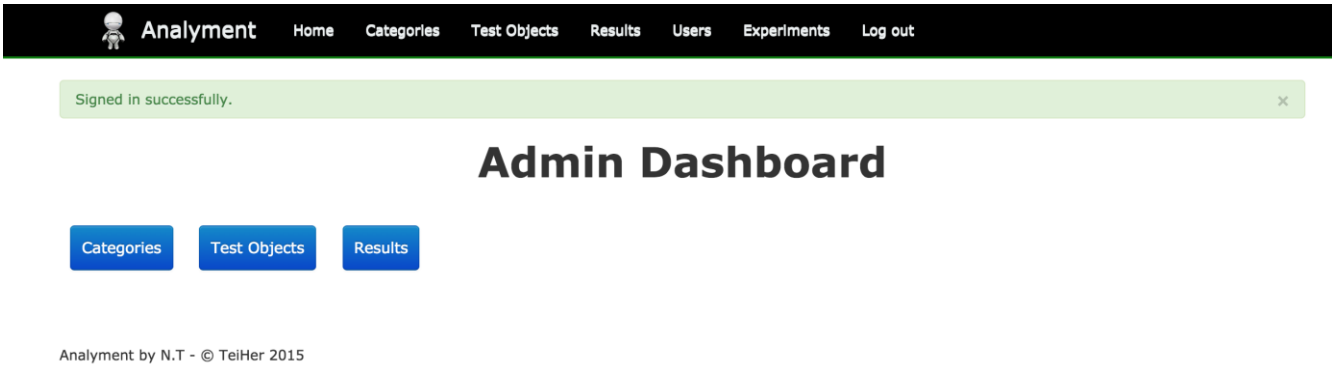
Εικόνα 5.4.1-3: Λειτουργία ανάκτησης κωδικού

5.4.2 Λειτουργίες διαχειριστών - απλών χρηστών

Στην εφαρμογή Analyment οπως και σε όλες τις διαδικτυακές εφαρμογές υπάρχουν δύο είδη χρηστών, απλός χρήστης και ο διαχειριστής. Εδώ ο απλός χρήστης τα μόνα δικαιώματα που έχει είναι αυτό της αξιολόγισης και της προβολής των αποτελεσμάτων αφού έχει τελειώσει ολόκληρη την κατηγορία με τα κείμενα. Ο διαχειριστής από την άλλη έχει όλα τα δικαιώματα ακόμα και να κάνει τον απλό χρήσει διαχειριστή και το αντίθετο.

Αρχική σελίδα διαχειριστή

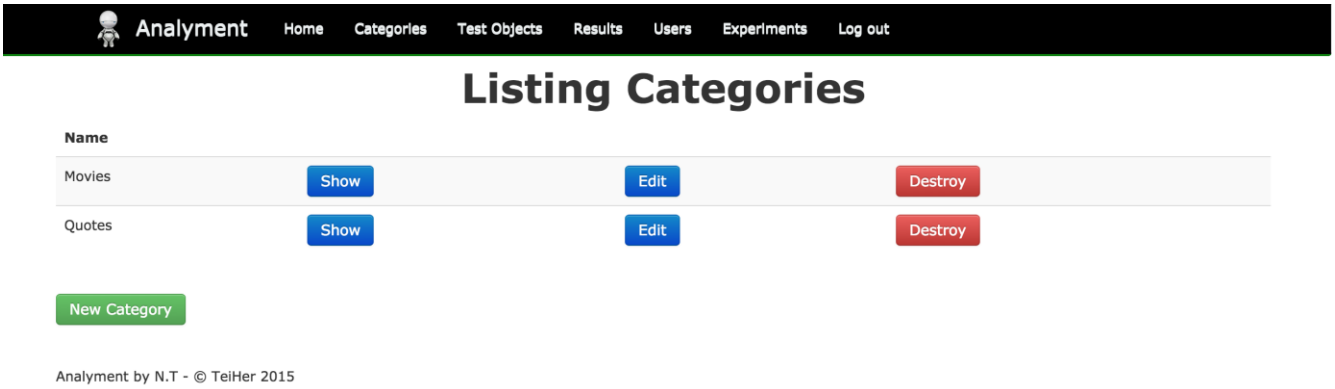
Η πρώτη σελίδα περιέχει μόνο ένα γρήγορο μενού που οδηγεί σε όλες τις λειτουργίες, και υπάρχει σε όλες τις σελίδες για να κάνει την πλοήγιση πιο εύκολη.



Εικόνα 5.4.2-1: Αρχική σελίδα admin

Categories-Κατηγορίες

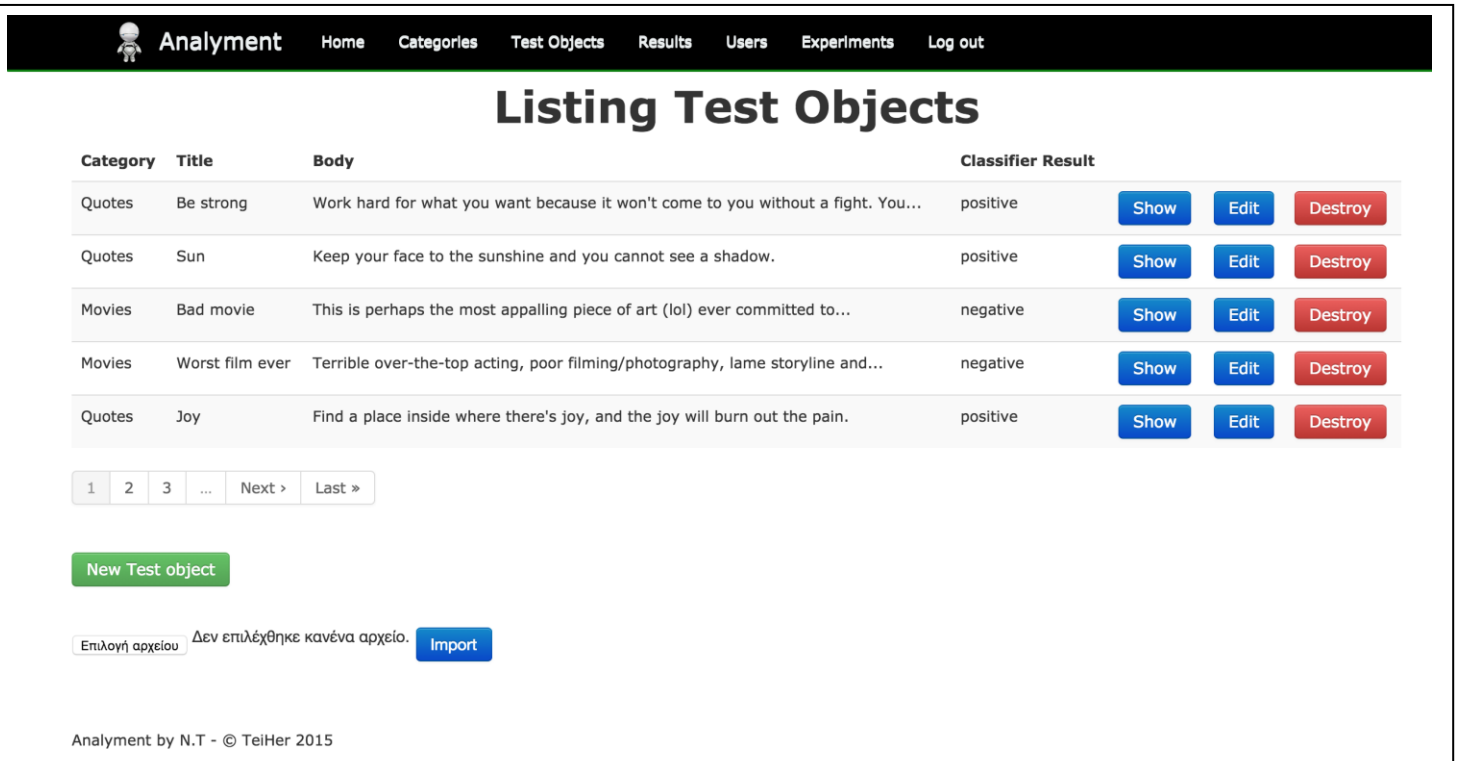
Στο κουμπί Categories φαίνονται όλες οι κατηγορίες που υπάρχουν για “βαθμολόγιση”. Ακόμα δίνεται η δυνατότητα να επεξεργαστούν να δημιουργηθούν νέες αλλά και να διαγραφούν μαζί με όλα τα κείμενα τους.



Εικόνα 5.4.2-2: ‘Όλες οι κατηγορίες των πειραμάτων

Αντικείμενα πρός βαθμολόγιση

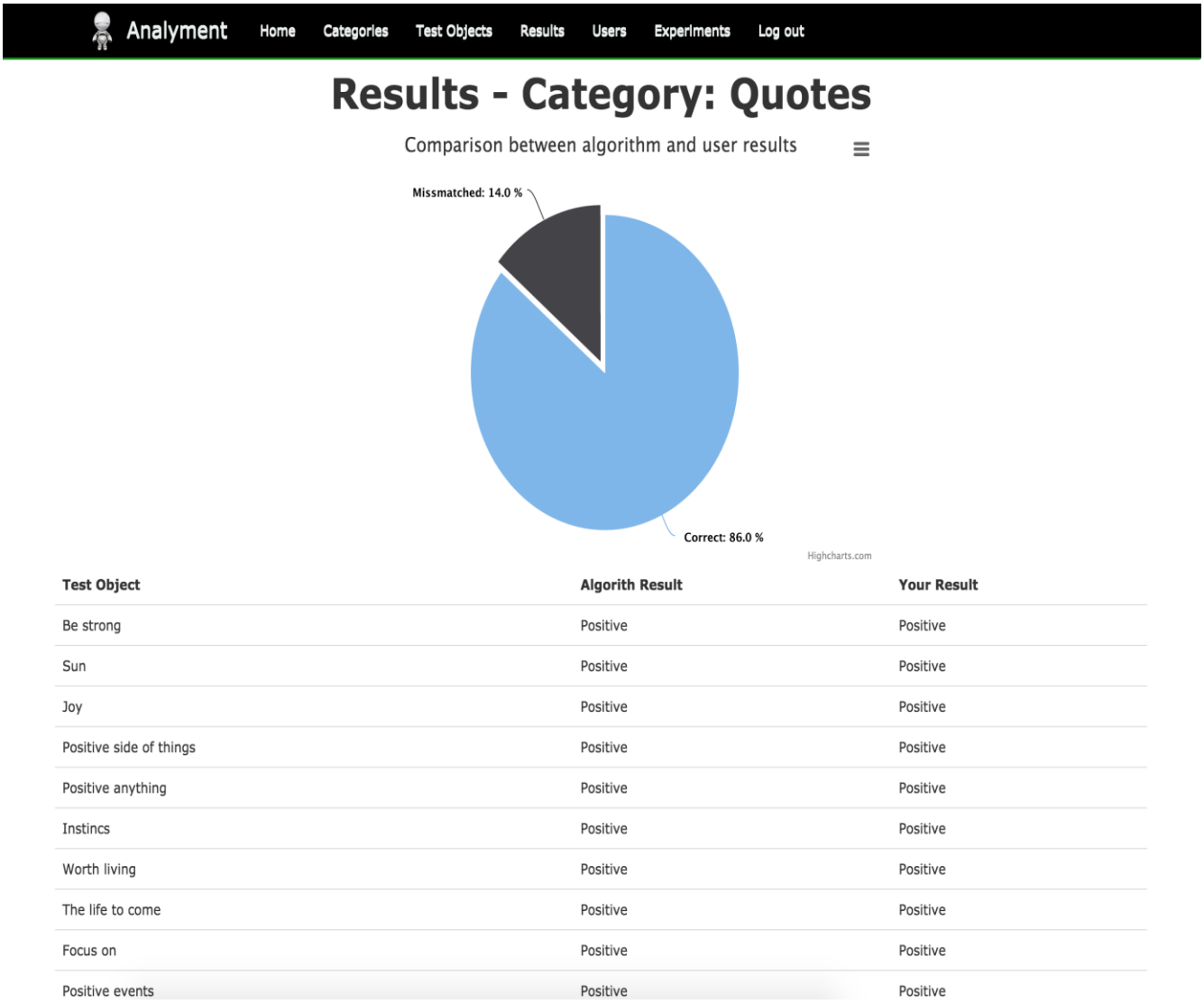
Σε αυτό το σημείο είναι συγκεντρωμένα όλα τα κείμενα που είναι πρός κατηγοριοποίηση. Εδώ μπορεί ο διαχειρηστής να εισάγει νέα κείμενα είτε μαζικά από ένα csv αρχείο είτε ένα-ένα χειροκίνητα. Ακόμα μπορεί το κάθε κείμενο να επεξεργαστεί και να διαγραφεί. Όπως είναι φυσικό τα κείμενα μίας κατηγορίας μπορεί να είναι πολλά και να μην αρκεί μια σελίδα να εμφανιστούν όλα, για αυτό προστεθηκε σελιδοποίηση στην εφαρμοφη για να τα ταξινομεί ανά ομάδες κειμένων.



Εικόνα 5.4.2-3: Σελίδα διαχείρησης των Κειμένων

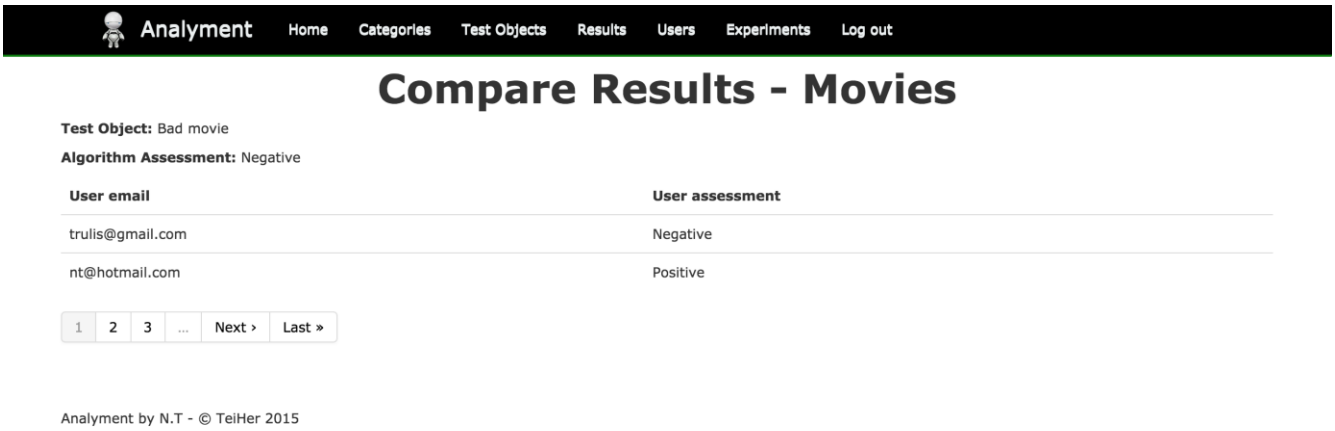
Compare Results

Η σελίδα με τα συγκεντρωτικά αποτελέσματα των χρηστών. Μπορεί να την δεί μόνο ο διαχειρηστής καθώς οι χρήστες μπορούν να δούν μόνο τα δικά τους.



Εικόνα 5.4.2-4: Σύγκριση αποτελεσμάτων ανά κατηγορία

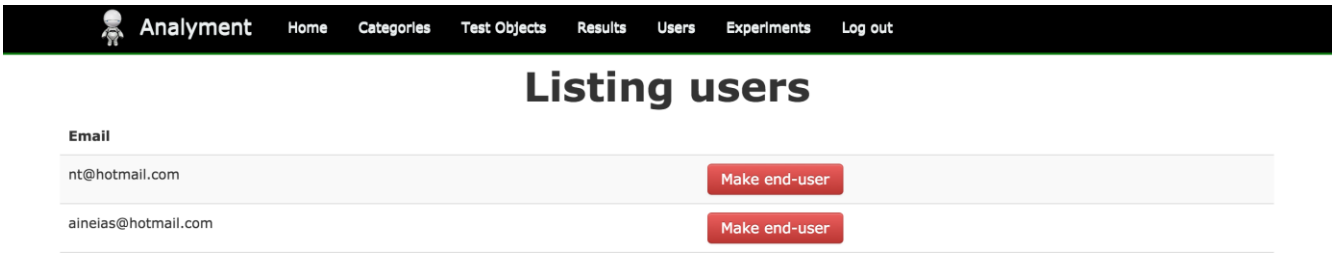
Αν επιλέξει μία κατηγορία του εμφανίζονται σε κάθε σελίδα τα κείμενά της καθώς επίσης την βαθμολογία του αλγορίθμου, τα email των χρηστών και πώς το βαθμολόγησε ο κάθε χρήστης το συγκεκριμένο κείμενο.



Εικόνα 5.4.2-5: Σελίδα σύγκρισης αποτελεσμάτων ανά Test Object

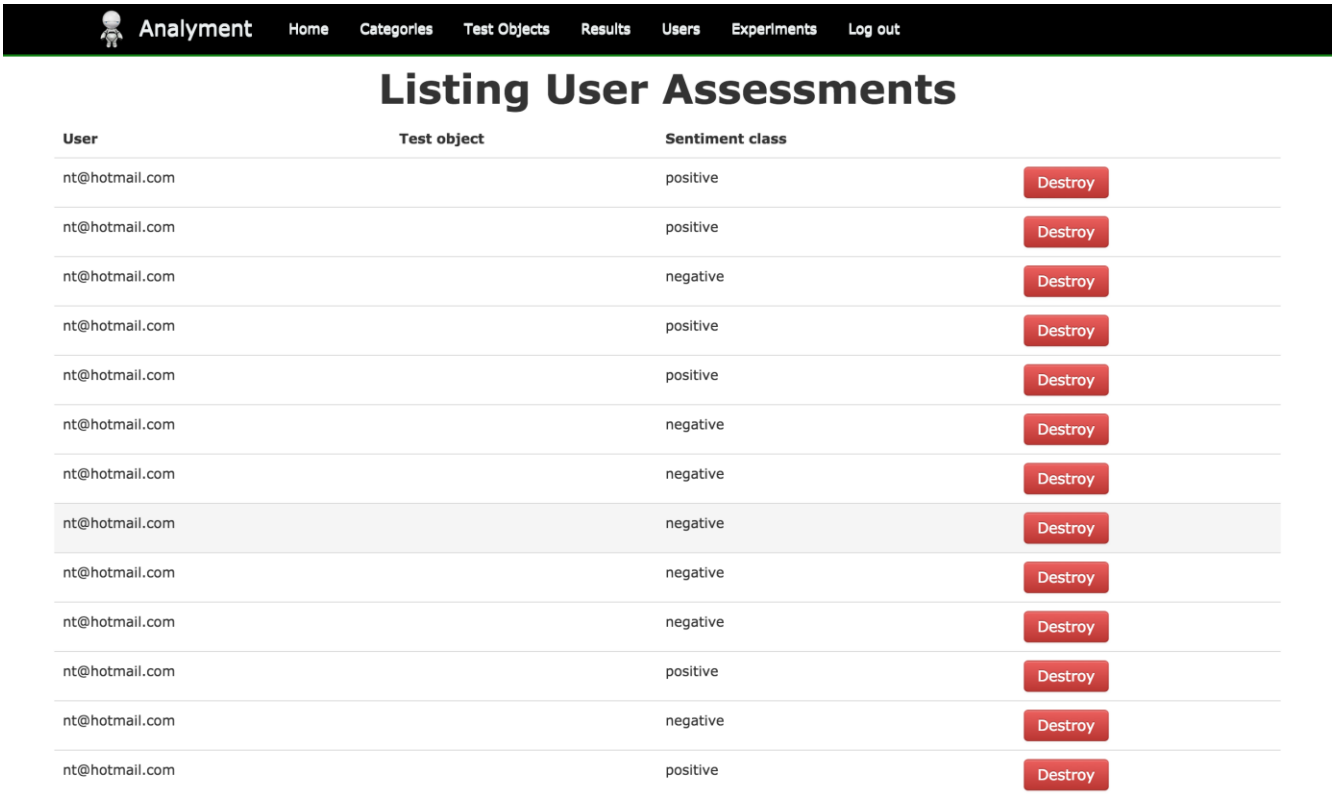
Χρήστες

Σε περίπτωση που υπάρξουν πολλοί χρήστες προστέθηκε η λειτουργία στην εφαρμογή όλοι οι χρήστες να μπορούν να γίνουν διαχειριστές καθώς και το αντίστροφο, δηλαδή ένας διαχειριστής να μπορεί να κάνει τον εαυτό του απλό χρήστη.



Εικόνα 5.4.2-6: Σελίδα χρηστών

Με το πράσινο κουμπί ο διαχειριστής μπορεί να δεί συγκεντρωτικά τα αποτελέσματα ακόμα και να τα διαγραψει όταν έχει ολοκληρωθεί η πειραματική διαδικασία.

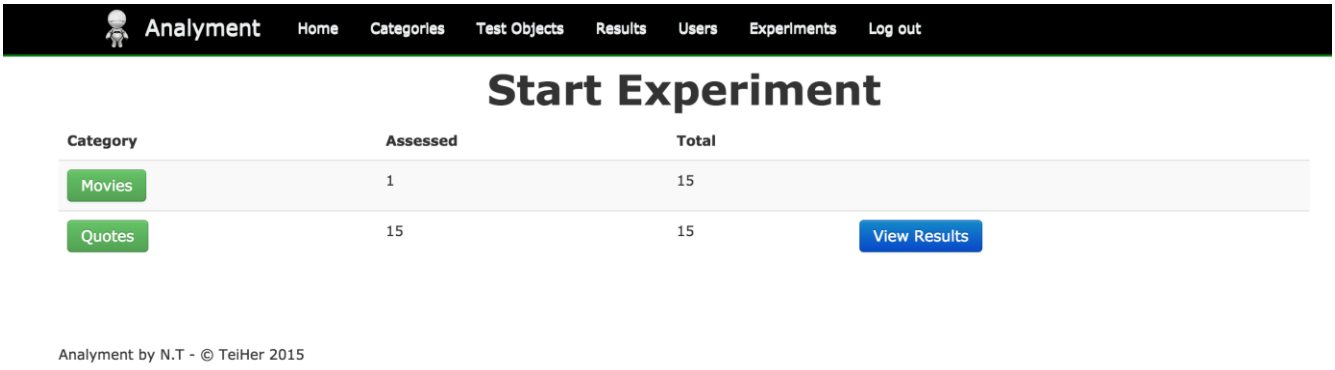


Εικόνα 5.4.2-7: Εμφάνιση όλων τών κειμένων

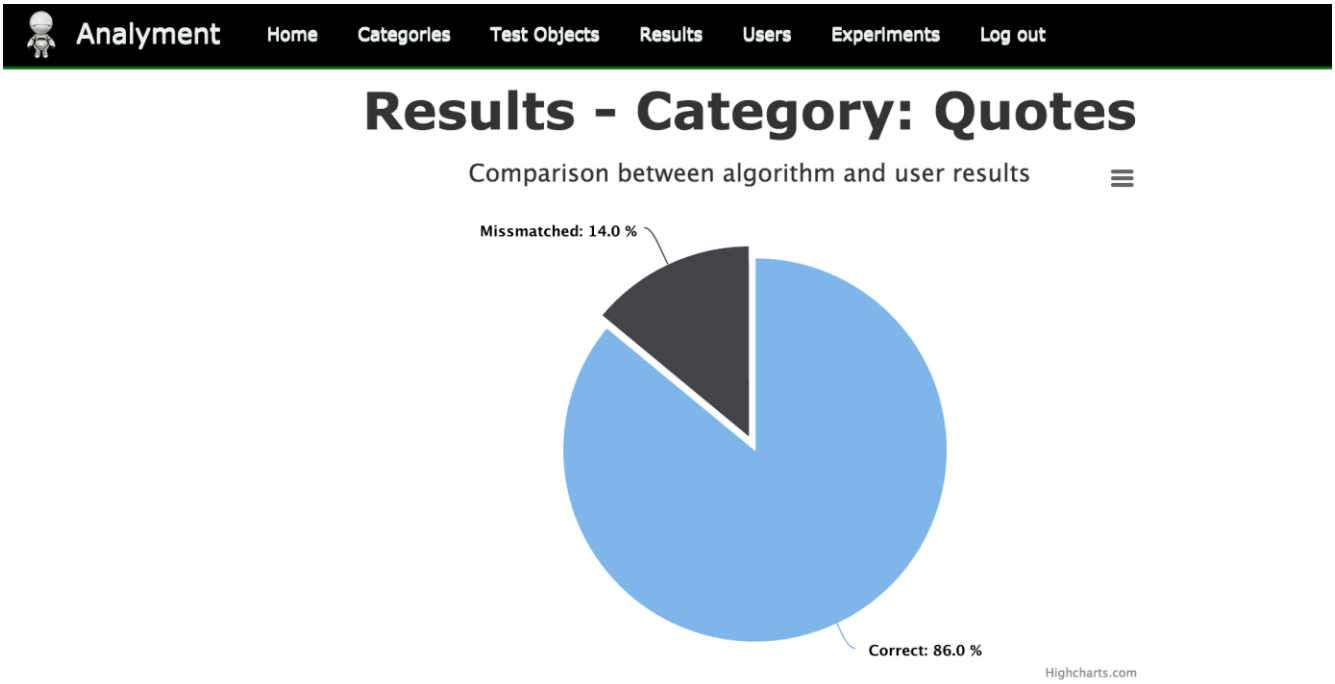
Πειράματα

Η τελευταία και βασική λειτουργία της εφαρμογής είναι το Experiment, η οποία είναι και η μοναδική κοινή λειτουργία των διαχειριστών με τους απλούς χρήστες.

Εδώ παρουσιάζονται όλα τα πειράματα και ο συνολικός αριθμός των κειμένων που περιέχουν και πόσα απο αυτά ο χρήστης έχει βαθμολογήσει. Όταν κατηγοριοποιήσει όλα τα κείμεναεμφανίζεται ένα κουμπί που οδηγέι στην σελίδα των αποτελεσμάτων μαζί με ένα γράφημα δείχοντας τι ποσοστο απόκλισης είχε ο αλγόριθμος από τον χρήστη.



Εικόνα 5.4.2-8: Πειραματική διαδικασία για τον διαχειριστή



Εικόνα 5.4.2-9: Γράφημα με την ολοκλήρωση της πειραματικής διαδικασίας

Εικόνα 5.4.2-10: Ανάθεση βαθμολογίας στα κείμενα

5.5 Προβλήματα - Συμπεράσματα και μελλοντική εξέλιξη της εφαρμογής

Αυτό το κεφάλαιο αφιερώνεται στην καταγραφή των προβλημάτων που αντιμετώπισε η ομάδα μας καθώς και τον τρόπο που έφτασε στην επίλυσή τους. Αξίζει να αναφερθεί ότι και κατά την σχεδίαση και κατά την υλοποίηση της εφαρμογής Analyment η αναζήτηση στα κατάλληλα φόρουμ στο διαδίκτυο μας οδήγησε στην εύρεση της λύσης, καθώς όλοι οι προγραμματιστές έχουν αντιμετωπίσει κάποιο πρόβλημα και το έχουν μοιραστεί στα ειδικά φόρουμ.

Το πρώτο πρόβλημα που αντιμετωπίστηκε ήταν η εγκατάσταση του περιβάλλοντος εργασίας της γλώσσας προγραμματισμού Ruby αλλά και του framework της Ruby on Rails, καθώς ήταν απαραίτητη η εργασία απο διαφορετικά λειτουργικά συστήματα. Χρησιμοποιήθηκαν οι εκδόσεις λειτουργικών Ubuntu 12.04 και Mac OSX 10.10. Μετά από αναζήτηση στο διαδίκτυο (κυρίως στο stackoverflow.com) βρέθηκε τρόπος εγκατάστασης της πλατφόρμας στα διαφορετικά λειτουργικά συστήματα. Ο λόγος ύπαρξής τους ήταν για να δοκιμάσουμε πώς λειτουργεί η εφαρμογή αλλά και επειδή η ομάδα είχε διαφορετικά λειτουργικά συστήματα.

Αναφέρθηκε σε προηγούμενο κεφάλαιο η ευκολία χρήσης του twitter bootstrap. Αξίζει να αναφέρουμε οτι αντιμετωπίστηκαν αρκετά προβλήματα κατά την εγκατάστασή της βιβλιοθήκης μέχρι να καταλήξουμε στην πιο λειτουργική, η οποία προτεινόταν και στα εγχειρίδια της βιβλιοθήκης

και της κοινότητας της Ruby on Rails.

Μια πολύ χρήσιμη βιβλιοθήκη, η kaminari βρέθηκε για να μας βοηθήσει στην αντιμετώπιση του προβλήματος της σελιδοποίησης. Λόγω του όγκου δεδομένων που έχει η εφαρμογή έπρεπε να τα ταξινομίσουμε για να είναι πιο εύκολα στην αναζήτηση αλλά και την πλοήγηση για τον χρήστη.

Μία από της πίο χρήσιμες βιβλιοθήκες είναι η device που διαχωρίζει δικαιώματα μεταξύ απλού χρήστη και διαχειριστη, ηταν δύσκολη η απόφαση με ποίον τρόπο θα διαχωριστούν τα δικαιώματα μέχρι την εγκατάστασή της. Επρεπε να την χρησιμοποιήσουμε αφού η εφαρμογή βασίζεται κυρίως στις λειτουργίες του διαχειριστή.

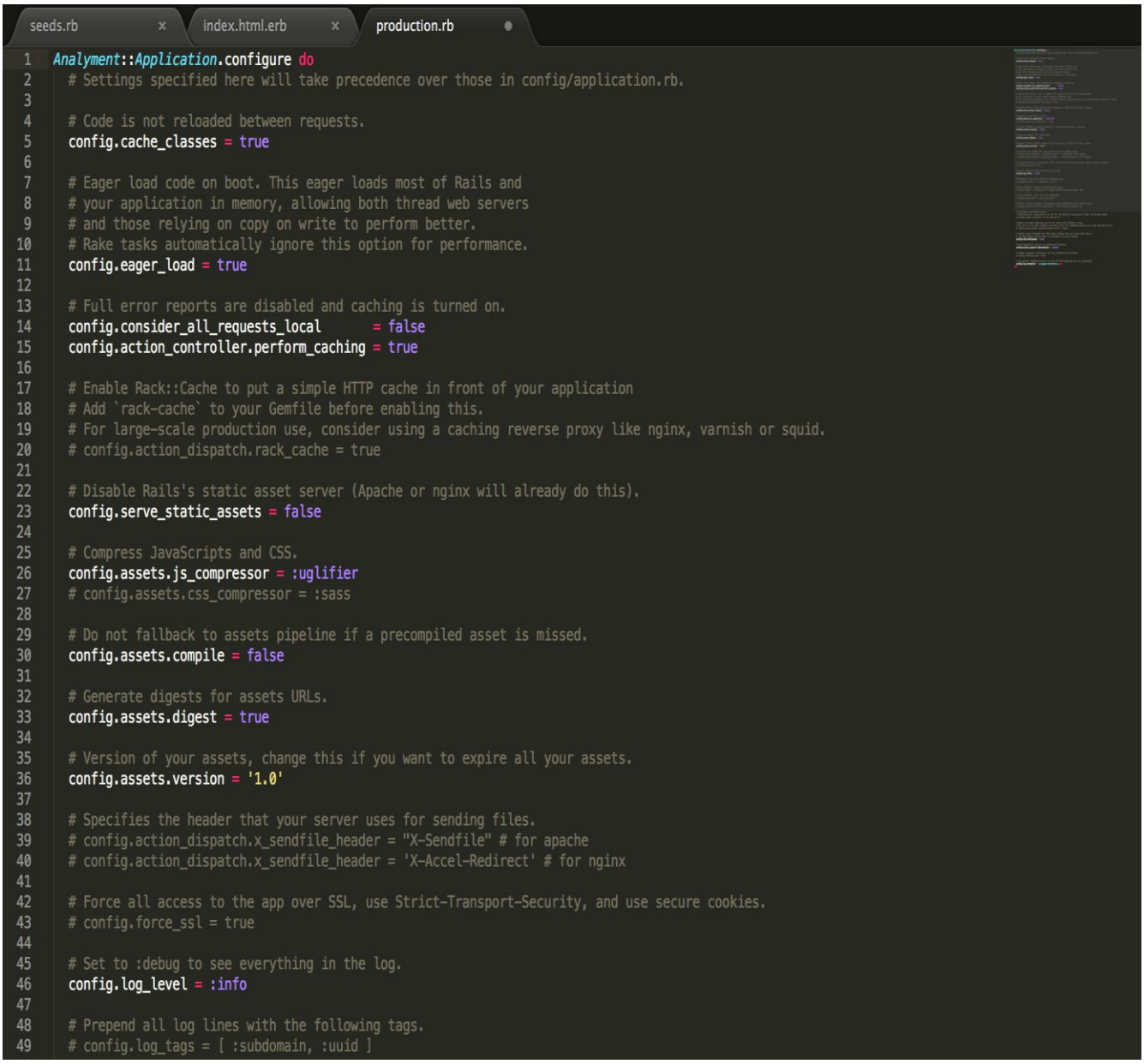
Το πίο δύσκολο πρόβλημα που αντιμετωπίστηκε ειναι αυτό του deployment στο υπολογιστικό νέφος το οποίο προκαλούσε δυσλειτουργία στην εφαρμογή. Η εφαρμογή για να δώσει τα επιθυμητά αποτελέσματα, εκτός από τα αρχεία που περιέχουν κώδικα σε Ruby έχει και αρχεία με κώδικα άλλων γλωσσών όπως javascript, CSS και εικόνες τα assets. Τα αρχεία αυτά περνάνε από σύνταξη (compilation) για να τρέξουν παράλληλα με τον κύριο κορμό της εφαρμογής και δημιουργήθηκαν προβλήματα στην σωστή λειτουργία της εφαρμογής.

Οι χρήστες μπορούν να κάνουν χειροκίνητη αξιολόγιση του συναισθήματος των κειμένων βοηθώντας την τελική εξαγωγή συμπεράσματων για την καλή λειτουργία των classifiers. Εάν και καταλήξαμε στην υλοποίηση ενός εύχρηστου και ευκολου “εργαλείου” ο σκοπός είναι το εξελίξουμε στο μέλλον σε μία ποιο αυτοματοποιημένη εφαρμογή. Το πρώτο βήμα είναι να είναι συνδεδεμένος ο classifier με την εφαρμογή και να κάνει real time εκπαίδευση. Αφού εκπαιδευτεί να εισάγει ο χρήστης URL της επιθυμίας του και ο αλγόριθμος να εμφανίζει το αποτέλεσμα. Η πιο χρήσιμη λειτουργία για τον αλγόριθμο είναι τα κείμενα τα οποία κατηγοριοποιεί σωστά να τα παίρνει ως ανατροφοδότιση “feedback” και να αποτελούν παραπάνω σύνολα εκπαίδευσης. Οι λειτουργίες αυτές δεν υλοποιήθηκαν λόγω έλειψης χρόνου και αυξημένων υποχρεώσεων, αλλά το βασικότερο λόγω αυξημένων προγραμματιστικών αναγκών.

Παράρτημα

Κατάλογος κώδικα

Production.rb


seeds.rb
index.html.erb
production.rb
Analyment::Application.configure do
# Settings specified here will take precedence over those in config/application.rb.
# Code is not reloaded between requests.
config.cache_classes = true
# Eager load code on boot. This eager loads most of Rails and
# your application in memory, allowing both thread web servers
# and those relying on copy on write to perform better.
# Rake tasks automatically ignore this option for performance.
config.eager_load = true
# Full error reports are disabled and caching is turned on.
config.consider_all_requests_local = false
config.action_controller.perform_caching = true
# Enable Rack::Cache to put a simple HTTP cache in front of your application
# Add `rack-cache` to your Gemfile before enabling this.
# For large-scale production use, consider using a caching reverse proxy like nginx, varnish or squid.
# config.action_dispatch.rack_cache = true
# Disable Rails's static asset server (Apache or nginx will already do this).
config.serve_static_assets = false
# Compress JavaScripts and CSS.
config.assets.js_compressor = :uglifier
# config.assets.css_compressor = :sass
# Do not fallback to assets pipeline if a precompiled asset is missed.
config.assets.compile = false
# Generate digests for assets URLs.
config.assets.digest = true
# Version of your assets, change this if you want to expire all your assets.
config.assets.version = '1.0'
# Specifies the header that your server uses for sending files.
# config.action_dispatch.x_sendfile_header = "X-Sendfile" # for apache
# config.action_dispatch.x_sendfile_header = 'X-Accel-Redirect' # for nginx
# Force all access to the app over SSL, use Strict-Transport-Security, and use secure cookies.
# config.force_ssl = true
# Set to :debug to see everything in the log.
config.log_level = :info
# Prepend all log lines with the following tags.
# config.log_tags = [:subdomain, :uuid]

```
48    # Prepend all log lines with the following tags.
49    # config.log_tags = [ :subdomain, :uuid ]
50
51    # Use a different logger for distributed setups.
52    # config.logger = ActiveSupport::TaggedLogging.new(SyslogLogger.new)
53
54    # Use a different cache store in production.
55    # config.cache_store = :mem_cache_store
56
57    # Enable serving of images, stylesheets, and JavaScripts from an asset server.
58    # config.action_controller.asset_host = "http://assets.example.com"
59
60    # Precompile additional assets.
61    # application.js, application.css, and all non-JS/CSS in app/assets folder are already added.
62    # config.assets.precompile += %w( search.js )
63
64    # Ignore bad email addresses and do not raise email delivery errors.
65    # Set this to true and configure the email server for immediate delivery to raise delivery errors.
66    # config.action_mailer.raise_delivery_errors = false
67
68    # Enable locale fallbacks for I18n (makes lookups for any locale fall back to
69    # the I18n.default_locale when a translation can not be found).
70    config.i18n.fallbacks = true
71
72    # Send deprecation notices to registered listeners.
73    config.active_support.deprecation = :notify
74
75    # Disable automatic flushing of the log to improve performance.
76    # config.autoflush_log = false
77
78    # Use default logging formatter so that PID and timestamp are not suppressed.
79    config.log_formatter = ::Logger::Formatter.new
80  end
81
```

Aplication.rb

Results.js.coffee

results.js.coffee

```
$ ->
  $("#comparison_chart").highcharts
    chart:
      plotBackgroundColor: null
      plotBorderWidth: null
      plotShadow: false

    title:
      text: "Comparison between algorithm and user results"

    tooltip:
      pointFormat: "{series.name}: <b>{point.percentage:.1f}%</b>"

    plotOptions:
      pie:
        allowPointSelect: true
        cursor: "pointer"
        dataLabels:
          enabled: true
          format: "<b>{point.name}</b>: {point.percentage:.1f} %"
          style:
            color: (Highcharts.theme and Highcharts.theme.contrastTextColor) or "black"

    series: [
      type: "pie"
      name: "User assessments"
      data: [
        {
          name: "Correct"
          y: parseInt($('#user_correct').text())
          sliced: true
          selected: true
        }
        [
          "Missmatched"
          100 - parseInt($('#user_correct').text())
        ]

      ]
    ]

  return

```

Custom.css.scss

custom.css.scss

```
body {
  background-image: white;
}

h1 {
  text-align: center
}

table {
  margin: 0 auto;
}

footer{
  margin-top: 40px;
}
.container-fluid {
background-color: Black;
}
.navbar {
overflow: hidden;
margin-bottom: 0px;
}
.main-wrapper {
  width: 1170px;
  margin: 0 auto;
  background-color: white;
}
.navbar .nav {
  margin-top:5px;
}
.navbar .brand:hover {
  background-color: #3E4449;
  text-decoration: none;
  color: #777777;
  text-shadow: 0 1px 0 #777777;
}

.navbar .brand {
float: left;
display: block;
padding: 8px 20px 4px;
margin-left: 50px;
font-size: 20px;
font-weight: 200;
color: #ecf0f1;
text-shadow: 0 1px 0 #ecf0f1;
}
.navbar-inner {

  padding-left: 0px;
  padding-right: 0px;
  background-color: #3E4449;
  background-image: -webkit-linear-gradient(top, #333333, #222222);
  border-bottom: 2px solid green;
}
```

```
56
57  .navbar .nav>li>a {
58  float: none;
59  padding: 10px 15px 10px;
60  color: #ecf0f1;
61  text-decoration: none;
62  text-shadow: 0 1px 0 #ecf0f1;
63  display: block;
64  }
65
66  .navbar .nav>li>a:hover {
67    color: #777777;
68    text-shadow: 0 1px 0 #777777;
69
70  }
71
72
73
74
75
76  .alert {
77    margin-top: 20px;
78  }
79
80  .dashboard {
81  }
82
83  .dashboard a {
84    margin: 30px 10px;
85    padding: 10px;
86  }
87
88  a.btn-primary:visited, a.btn-success:visited, a.btn-danger:visited {color:#fff}
89
90  .btn-default {
91    color: #523352;
92  }
93
94  h4.result_categories {
95    float: left;
96  }
97  .result_categories li {
98    list-style: none;
99    float: left;
100   margin: 5px 10px;
101 }
102
103
104 img.style_image{
105   margin: 20px auto;
106   display: block;
107   width: 75%;
108 }
```

assessments_controller.rb

user_assessments_controller.rb

```ruby
class UserAssessmentsController < ApplicationController
  authorize_resource class: false
  before_action :set_user_assessment, only: [:show, :edit, :update, :destroy]

  # GET /user_assessments
  # GET /user_assessments.json
  def index
    @user_assessments = UserAssessment.all.page(params[:page]).per(20)
  end

  # POST /user_assessments
  # POST /user_assessments.json
  def create
    @user_assessment = UserAssessment.new(user_assessment_params)
    @user_assessment.user_id = current_user.id
    set_experiment_validation_corpus
    respond_to do |format|
      if @user_assessment.save
        format.html { redirect_to :back, notice: 'User assessment was successfully created.' }
        format.json { render action: 'show', status: :created, location: @user_assessment }
      else
        if @user_assessment.errors[:user_id] == ["has already been taken"]
          update and return
        end
        format.html { render action: 'new' }
        format.json { render json: @user_assessment.errors, status: :unprocessable_entity }
      end
    end
  end

  def update
    @user_assessment = UserAssessment.where(user_id: current_user.id, test_object_id: user_assessment_params[:test_object_id] ).f
    respond_to do |format|
      if @user_assessment.update_attributes(sentiment_class_id: user_assessment_params[:sentiment_class_id])
        format.html { redirect_to :back, notice: 'User assessment was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: 'edit' }
        format.json { render json: @user_assessment.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /user_assessments/1
  # DELETE /user_assessments/1.json
  def destroy
    @user_assessment.destroy
    respond_to do |format|
      format.html { redirect_to user_assessments_url }
      format.json { head :no_content }
    end
  end

  def set_experiment_validation_corpus
    @user_assessment.experiment_validation_corpus_id = TestObject.find(user_assessment_params[:test_object_id]).experiment_valida
  end

```

```
57
58    private
59      # Use callbacks to share common setup or constraints between actions.
60      def set_user_assessment
61        @user_assessment = UserAssessment.find(params[:id])
62      end
63
64      # Never trust parameters from the scary internet, only allow the white list through.
65      def user_assessment_params
66        params.require(:user_assessment).permit(:user_id, :test_object_id, :sentiment_class_id)
67      end
68  end
69
```

objects_controller.rb

user_assessments_controller.rb × test_objects_controller.rb ×

```ruby
class TestObjectsController < ApplicationController
  authorize_resource class: false
  before_action :set_test_object, only: [:show, :edit, :update, :destroy]

  # GET /test_objects
  # GET /test_objects.json
  def index
    @test_objects = TestObject.all.page(params[:page]).per(5)
  end

  # GET /test_objects/1
  # GET /test_objects/1.json
  def show
  end

  # GET /test_objects/new
  def new
    @test_object = TestObject.new
  end

  # GET /test_objects/1/edit
  def edit
  end

  # POST /test_objects
  # POST /test_objects.json
  def create
    @test_object = TestObject.new(test_object_params)

    respond_to do |format|
      if @test_object.save
        format.html { redirect_to @test_object, notice: 'Test object was successfully created.' }
        format.json { render action: 'show', status: :created, location: @test_object }
      else
        format.html { render action: 'new' }
        format.json { render json: @test_object.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /test_objects/1
  # PATCH/PUT /test_objects/1.json
  def update
    respond_to do |format|
      if @test_object.update(test_object_params)
        format.html { redirect_to @test_object, notice: 'Test object was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: 'edit' }
        format.json { render json: @test_object.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /test_objects/1
  # DELETE /test_objects/1.json
```

```ruby
  # DELETE /test_objects/1.json
  def destroy
    @test_object.destroy
    respond_to do |format|
      format.html { redirect_to test_objects_url }
      format.json { head :no_content }
    end
  end

  def import_csv
    TestObject.import(params[:file])
    redirect_to test_objects_path, notice: "Test objects imported."
  end

  def import_body
    test_object = TestObject.find(params[:test_object_id])
    file_name = test_object.body
    file = File.open("#{Rails.root}/tmp/#{file_name}")
    test_object.body = file.read()
    test_object.save
    redirect_to(:back)
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_test_object
      @test_object = TestObject.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def test_object_params
      params.require(:test_object).permit(:experiment_validation_corpus_id, :title, :body, :sentiment_class_id)
    end
end

```

1 **classes_ controller.rb**

user_assessments_controller.rb | test_objects_controller.rb | sentiment_classes_controller.rb

```ruby
class SentimentClassesController < ApplicationController
  authorize_resource class: false
  before_action :set_sentiment_class, only: [:show, :edit, :update, :destroy]

  # GET /sentiment_classes
  # GET /sentiment_classes.json
  def index
    @sentiment_classes = SentimentClass.all
  end

  # GET /sentiment_classes/1
  # GET /sentiment_classes/1.json
  def show
  end

  # GET /sentiment_classes/new
  def new
    @sentiment_class = SentimentClass.new
  end

  # GET /sentiment_classes/1/edit
  def edit
  end

  # POST /sentiment_classes
  # POST /sentiment_classes.json
  def create
    @sentiment_class = SentimentClass.new(sentiment_class_params)

    respond_to do |format|
      if @sentiment_class.save
        format.html { redirect_to @sentiment_class, notice: 'Sentiment class was successfully created.' }
        format.json { render action: 'show', status: :created, location: @sentiment_class }
      else
        format.html { render action: 'new' }
        format.json { render json: @sentiment_class.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /sentiment_classes/1
  # PATCH/PUT /sentiment_classes/1.json
  def update
    respond_to do |format|
      if @sentiment_class.update(sentiment_class_params)
        format.html { redirect_to @sentiment_class, notice: 'Sentiment class was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: 'edit' }
        format.json { render json: @sentiment_class.errors, status: :unprocessable_entity }
      end
    end
  end
```

```ruby

  # DELETE /sentiment_classes/1
  # DELETE /sentiment_classes/1.json
  def destroy
    @sentiment_class.destroy
    respond_to do |format|
      format.html { redirect_to sentiment_classes_url }
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_sentiment_class
      @sentiment_class = SentimentClass.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def sentiment_class_params
      params.require(:sentiment_class).permit(:name)
    end
end
```

1 Experiments_ controller.rb

experiments_controller.rb

```ruby
class ExperimentsController < ApplicationController
  def index
    @experiments = []
    ExperimentValidationCorpus.all.each do |experiment|
      exp = {}
      exp[:name] = experiment.name
      exp[:assessed] = assessed_by_user(experiment).length
      exp[:total] = count_category_test_objects(experiment)
      @experiments << exp
    end
  end

  def assessed_by_user experiment
    UserAssessment.where(user_id: current_user.id, experiment_validation_corpus_id: experiment.id)
  end

  def count_category_test_objects experiment
    TestObject.where(experiment_validation_corpus_id: experiment.id).length
  end

  def results
    @experiment_validation_corpus = ExperimentValidationCorpus.find_by_name(params[:format])
    # in case someone tries to see results (if he knows the URL) without having assessed all test objects
    if assessed_by_user(@experiment_validation_corpus).length == count_category_test_objects(@experiment_validation_corpus)
      @results = UserAssessment.where(experiment_validation_corpus_id: @experiment_validation_corpus.id, user_id: current_user.id

      @score = UserAssessment.algorithm_comparison(@results, current_user)
    end
  end

  def run_assessment
    @experiment_validation_corpus = ExperimentValidationCorpus.find_by_name(params[:format])
    @test_objects = TestObject.where(experiment_validation_corpus_id: @experiment_validation_corpus.id).page(params[:page]).per(1
    @user_assessment = UserAssessment.new
  end
end

```

1 Validation_Corpora_controller.rb

```ruby
class ExperimentValidationCorporaController < ApplicationController
  authorize_resource class: false
  before_action :set_experiment_validation_corpus, only: [:show, :edit, :update, :destroy]

  # GET /experiment_validation_corpora
  # GET /experiment_validation_corpora.json
  def index
    @experiment_validation_corpora = ExperimentValidationCorpus.all
  end

  # GET /experiment_validation_corpora/1
  # GET /experiment_validation_corpora/1.json
  def show
  end

  # GET /experiment_validation_corpora/new
  def new
    @experiment_validation_corpus = ExperimentValidationCorpus.new
  end

  # GET /experiment_validation_corpora/1/edit
  def edit
  end

  # POST /experiment_validation_corpora
  # POST /experiment_validation_corpora.json
  def create
    @experiment_validation_corpus = ExperimentValidationCorpus.new(experiment_validation_corpus_params)

    respond_to do |format|
      if @experiment_validation_corpus.save
        format.html { redirect_to @experiment_validation_corpus, notice: 'Category was successfully created.' }
        format.json { render action: 'show', status: :created, location: @experiment_validation_corpus }
      else
        format.html { render action: 'new' }
        format.json { render json: @experiment_validation_corpus.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /experiment_validation_corpora/1
  # PATCH/PUT /experiment_validation_corpora/1.json
  def update
    respond_to do |format|
      if @experiment_validation_corpus.update(experiment_validation_corpus_params)
        format.html { redirect_to @experiment_validation_corpus, notice: 'Category was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render action: 'edit' }
        format.json { render json: @experiment_validation_corpus.errors, status: :unprocessable_entity }
      end
    end
  end

```

```ruby

  # DELETE /experiment_validation_corpora/1
  # DELETE /experiment_validation_corpora/1.json
  def destroy
    @experiment_validation_corpus.destroy
    respond_to do |format|
      format.html { redirect_to experiment_validation_corpora_url }
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_experiment_validation_corpus
      @experiment_validation_corpus = ExperimentValidationCorpus.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def experiment_validation_corpus_params
      params.require(:experiment_validation_corpus).permit(:name)
    end
end

```

File – H:\Analyment\ Dashboards_ controller.rb

```ruby
class DashboardsController < ApplicationController
  authorize_resource class: false
  def admin_home
  end

  def index
    @experiment_validation_corpora = ExperimentValidationCorpus.all
  end

  def compare
    @experiment_validation_corpus = ExperimentValidationCorpus.find(params[:format])
    @test_objects_by_corpus = @experiment_validation_corpus.test_objects.page(params[:page]).per(1)
  end

  def users_index
    @users = User.all
  end

  def user_role
    @user = User.find(params[:format])
    if @user.role.present?
      @user.update_attributes(role: "")
    else
      @user.update_attributes(role: "admin")
    end
    redirect_to :back, notice: 'User role was successfully updated.'
  end
end
```

13

14 **File – H:\Analyment\ application.html.erb**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= content_for?(:title) ? yield(:title) : "Analyment" %></title>
    <%= csrf_meta_tags %>

    <!-- Le HTML5 shim, for IE6-8 support of HTML elements -->
    <!--[if lt IE 9]>
      <script src="//cdnjs.cloudflare.com/ajax/libs/html5shiv/3.6.1/html5shiv.js" type="text/javascript"></script>
    <![endif]-->

    <%= stylesheet_link_tag "application", :media => "all" %>

    <!-- For third-generation iPad with high-resolution Retina display: -->
    <!-- Size should be 144 x 144 pixels -->
    <%= favicon_link_tag 'apple-touch-icon-144x144-precomposed.png', :rel => 'apple-touch-icon-precomposed', :type => 'image/png'
      , :sizes => '144x144' %>

    <!-- For iPhone with high-resolution Retina display: -->
    <!-- Size should be 114 x 114 pixels -->
    <%= favicon_link_tag 'apple-touch-icon-114x114-precomposed.png', :rel => 'apple-touch-icon-precomposed', :type => 'image/png'
      , :sizes => '114x114' %>

    <!-- For first- and second-generation iPad: -->
    <!-- Size should be 72 x 72 pixels -->
    <%= favicon_link_tag 'apple-touch-icon-72x72-precomposed.png', :rel => 'apple-touch-icon-precomposed', :type => 'image/png',
      :sizes => '72x72' %>

    <!-- For non-Retina iPhone, iPod Touch, and Android 2.1+ devices: -->
    <!-- Size should be 57 x 57 pixels -->
    <%= favicon_link_tag 'apple-touch-icon-precomposed.png', :rel => 'apple-touch-icon-precomposed', :type => 'image/png' %>

    <!-- For all other devices -->
    <!-- Size should be 32 x 32 pixels -->
    <%= favicon_link_tag 'favicon.ico', :rel => 'shortcut icon' %>

    <%= javascript_include_tag "application" %>
    <script src="http://code.highcharts.com/highcharts.js"></script>
    <script src="http://code.highcharts.com/modules/exporting.js"></script>
  </head>
  <body>

    <div class="navbar navbar-fluid-top">
      <div class="navbar-inner">
        <div class="container-fluid">
          <a class="btn btn-navbar" data-target=".nav-collapse" data-toggle="collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </a>
          <a class="brand" href="/"><%= image_tag("logo1.png", size: "38") %> Analyment</a>
          <div class="container-fluid nav-collapse">
            <ul class="nav">
              <%= render '/layouts/menu' if current_user %>
```

15

```
55                </ul>
56              </div><!--/.nav-collapse -->
57            </div>
58          </div>
59        </div>
60
61        <div class="container-fluid main-wrapper">
62          <div class="row-fluid">
63            <div class="span12">
64              <%= bootstrap_flash %>
65              <%= yield %>
66            </div>
67          </div><!--/row-->
68
69          <footer>
70            <p>Analyment by N.T - &copy; TeiHer 2015</p>
71          </footer>
72
73        </div> <!-- /container -->
74
75      </body>
76    </html>
77
```

File – H:\Analyment\ schema.rb

```ruby
# encoding: UTF-8
# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# Note that this schema.rb definition is the authoritative source for your
# database schema. If you need to create the application database on another
# system, you should be using db:schema:load, not running all the migrations
# from scratch. The latter is a flawed and unsustainable approach (the more migrations
# you'll amass, the slower it'll run and the greater likelihood for issues).
#
# It's strongly recommended that you check this file into your version control system.

ActiveRecord::Schema.define(version: 20140111133824) do

  create_table "experiment_validation_corpora", force: true do |t|
    t.string   "name"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "sentiment_classes", force: true do |t|
    t.string   "name"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "test_objects", force: true do |t|
    t.integer  "experiment_validation_corpus_id"
    t.string   "title"
    t.text     "body"
    t.integer  "sentiment_class_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "test_objects", ["experiment_validation_corpus_id"], name: "index_test_objects_on_experiment_validation_corpus_id", u
  add_index "test_objects", ["sentiment_class_id"], name: "index_test_objects_on_sentiment_class_id", using: :btree

  create_table "user_assessments", force: true do |t|
    t.integer  "user_id"
    t.integer  "test_object_id"
    t.integer  "sentiment_class_id"
    t.integer  "experiment_validation_corpus_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "user_assessments", ["experiment_validation_corpus_id"], name: "index_user_assessments_on_experiment_validation_corpu
  add_index "user_assessments", ["sentiment_class_id"], name: "index_user_assessments_on_sentiment_class_id", using: :btree
  add_index "user_assessments", ["test_object_id"], name: "index_user_assessments_on_test_object_id", using: :btree
  add_index "user_assessments", ["user_id", "test_object_id"], name: "index_user_assessments_on_user_id_and_test_object_id", uniq
  add_index "user_assessments", ["user_id"], name: "index_user_assessments_on_user_id", using: :btree

  create_table "users", force: true do |t|
    t.string   "email",                  default: "", null: false
    t.string   "encrypted_password",     default: "", null: false
```

```
    t.string   "encrypted_password",     default: "", null: false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          default: 0,  null: false
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
    t.string   "role"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  add_index "users", ["email"], name: "index_users_on_email", unique: true, using: :btree
  add_index "users", ["reset_password_token"], name: "index_users_on_reset_password_token", unique: true, using: :btree

end
```

33

Αναφορές

[1]: David Thomas, Andrew Hunt, (2001), *The Pragmatic Programmer's Guide*. Διαθέσιμο: www.ruby-doc.org/docs/ProgrammingRuby/html/index.html. Προσπελάστηκε: 20η Απρ. 2012

[2]: Ruby on Rails community, (2014), *Rails Guides*. Διαθέσιμο: guides.rubyonrails.org/generators.html. Προσπελάστηκε: 1η Απρ. 2014

[3]: Oracle Corporation, (2014), MySQL. Διαθέσιμο: www.mysql.com/why-mysql. Προσπελάστηκε: 26η Μαρτίου 2014

[4]: Christian Neukirchen, (2012), Rack: a Ruby Webserver Interface. Διαθέσιμο: rack.github.com. Προσπελάστηκε: 26 Απρ. 2012

[5]: Michael Hartl, (2012), *Ruby on Rails Tutorial - Learn Rails by Example*. Διαθέσιμο: ruby.railstutorial.org/chapters. Προσπελάστηκε: 21η Απρ. 2012

[6]: Ruby on Rails community, (2012), *Rails Guides*. Διαθέσιμο: guides.rubyonrails.org/generators.html. Προσπελάστηκε: 21η Απρ. 2012

[7]: heroku, (2014), *heroku website*. Διαθέσιμο: www.heroku.com. Προσπελάστηκε: 25η Μαΐου 2014

[8]: C.Baun et al. (2011), Cloud Computing. 2nd ed. London: Springer. Κεφάλαιο 6 pp. 49-62

[9]: Roy T. Fielding, (2008), *REST APIs must be hypertext-driven*. Διαθέσιμο: roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. Προσπελάστηκε: 21η Απρ. 2014

[10]: Ruby on Rails Community, (2014), *Active Record Migrations*. Διαθέσιμο: http://api.rubyonrails.org/classes/ActiveRecord/Migration.htmlΠροσπελάστηκε: 28η Μαΐου 2014

[11]: http://www.cs.cornell.edu/people/pabo/movie-review-data

[12]: What is HTML?, (2014), Διαθέσιμο: www.yourhtmlsource.com. Προσπελάστηκε: 25η Μαΐου 2014

[13]: RVM, (2014), Διαθέσιμο: rvm.io. Προσπελάστηκε: 25η Μαΐου 2014

Βιβλιογραφία

[1] GAMON, M. 2004. Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis. In Proceedings the 20th International Conference on Computational Linguistics, pp. 841–847.
[2] Pang, B., & Lee, L. (2004, July). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In Proceedings of the 42nd annual meeting on Association for Computational Linguistics (p. 271). Association for Computational Linguistics.
[3] Xiaowen Ding, Bing Liu and Philip S. Yu. "A Holistic Lexicon-Based Appraoch to Opinion Mining." Proceedings of First ACM International Conference on Web Search and Data Mining (WSDM-2008), Feb 11-12, 2008, Stanford University, Stanford, California, USA
[4] SES: Sentiment Elicitation System for Social Media Data. Kunpeng Zhang, Yu Cheng, Yusheng Xie, Daniel Honbo, Ankit Agrawal, Diana Palsetia, Kathy Lee, Wei-keng Liao, Alok Choudhary. s.l. : 11th IEEE International Conference on Data Mining Workshops, 2011.
[5] Albert Bifet, Eibe Frank. Sentiment Knowledge Discovery in Twitter Streaming Data. Lecture Notes in Computer Science, Volume 6332, Discovery Science, Pages 1-15. 2010.
[6] Richard D. Waters, Jia Y. Jamal. Tweet, tweet, tweet: A content analysis of nonprofit organizations' Twitter updates. Public Relations Review. Elsevier Inc, September 2011, Τόμ. Volume 37, Issue 3, Pages 321-324.
[7] Farah Benamara, Baptiste Chardon, Yvette Yannick Mathieu, Vladimir Popescu: Towards Context-Based Subjectivity Analysis. IJCNLP 2011: 1180-1188
[8] Thorsten Joachims, Transductive Inference for Text Classification using Support Vector Machines. International Conference on Machine Learning (ICML), 1999.
[9] RAVI PARIKH - MATIN MOVASSATE, SENTIMENT ANALYSIS OF USERGENERATED TWITTER UPDATES USING VARIOUS CLASSICATION TECHNIQUES, JUNE 2009, STANDFORD UNIVERSITY
[10] An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Nello Cristianini and John Shawe-Taylor, Cambridge University Press, 2000
[11] V. PANDEY, K. IYER, SENTIMENT ANALYSIS OF MICROBLOGS, STANDFORD UNIVERSITY
[12] PREM MELVILLE - WOJCIECH GRYC - RICHARD D. LAWRENCE , SENTIMENT ANALYSIS OF BLOGS BY COMBINING LEXICAL KNOWLEDGE WITH TEXT CLASSIFICATION, KDD JUNE 2009
[13]Matsumoto, S., Takamura, H., & Okumura, M. (2005). Sentiment classification using word sub-sequences and dependency sub-trees. In Advances in Knowledge Discovery and Data Mining (pp.301-311). Springer Berlin Heidelberg.
[14]Kennedy, A., & Inkpen, D. (2006). Sentiment classification of movie reviews using contextual valence shifters. Computational Intelligence, 22(2), 110-125.
[15] Qiu, L., Zhang, W., Hu, C., & Zhao, K. (2009, November). Selc: a self-supervised model for sentiment classification. In Proceedings of the 18th ACM conference on Information and knowledge management (pp. 929-936). ACM.
[16] Hatzivassiloglou, V., Klavans, J. L., Holcombe, M. L., Barzilay, R., Kan, M. Y., & McKeown, K. (2001). Simfinder: A flexible clustering tool for summarization. Proceedings of the NAACL Workshop on Au-

tomatic Summarizatio.