README

Αντώνης Σκαρλάτος sdi1400184

WordCloud

Ο κώδικας του wordcloud περιέχεται στο αρχείο 'wordCloud.py'. Από τα κείμενα γίνεται αφαίρεση των stopwords και όλες τις λέξεις τις τροποποιούμε ώστε να μην περιέχουν κεφαλαία γράμματα. Επίσης μέσα στο wordcloud εμφανίζονται και οι λέξεις που υπάρχουν στους τίτλους. Τα αποτελέσματα αποθηκεύονται ως αρχεία .png στον φάκελο static

Χειρισμός κειμένων

Για κάθε κείμενο κρατάμε ένα text το οποίο περιέχει το content του κειμένου και τον τίτλο. Για τον λόγο ότι ο τίτλος προσδιορίζει αρκετά το περιεχόμενο ενός κειμένου, δεν εισάγουμε τον τίτλο μόνο μία φορά αλλά πολλές, ώστε να 'δυναμώσουμε' τις λέξεις που περιέχει ο τίτλος σε σχέση με τις υπόλοιπες. Μετά από δοκιμές, τα καλύτερα αποτελέσματα τα πήραμε όταν βάλαμε τον τίτλο $\frac{len(content)}{200}$ φορές. Στην συνέχεια μετατρέπεται το text έτσι ώστε να μην περιέχει κεφαλαία γράμματα, ώστε να καταλαβαίνει ο classifier την λέξη ως ίδια, είτε γράφεται με μικρά είτε με κεφαλαία γράμματα. Αφού ολοκληρωθούν τα παραπάνω βήματα, περνάμε το text στον vectorizer.

Vectorizers

Οι δύο vectorizers που δοκιμάστηκαν ήταν ο CountVectorizer και ο TfidfVectorizer. Ο καθένας δούλευε καλύτερα για κάποιους συγκεκριμένους classifiers όπως εξηγείται καλύτερα πιο κάτω. Και στους δύο vectorizers χρησιμοποιήθηκε η παράμετρος tokenizer, η οποία παίρνει σαν τιμή την συνάρτηση processText.

Στην συνάρτηση αυτή γίνεται κάποια επεξεργασία στα κείμενα. Πιο συγκεκριμένα η συνάρτηση βγάζει τα stopwords από τα κείμενα και στην συνέχεια εφαρμόζει πάνω σε αυτά πρώτα lemmatization για να πάρουμε την κοινή 'ρίζα' και στην συνέχεια stemming για να 'κόψουμε' την λέξη. Στους vectorizers δοκιμάστηκαν διάφορες παραμέτρους. Μία από αυτές ήταν η ngram_range=(1, 2), η οποία 'ενώνει' δύο λέξεις. Παρόλα αυτά όμως πήραμε καλύτερα αποτελέσματα χωρίς αυτήν την παράμετρο. Επίσης δοκιμάστηκαν οι παράμετροι max_df και min_df, χωρίς όμως να μας δίνουν καλύτερα αποτελέσματα.

TruncatedSVD

Χρησιμοποιήθηκε truncation ώστε να μικρύνουμε τις διαστάσεις και να γίνει το classification πιο γρήγορο και πιο αποδοτικό. Στον φάκελο static υπάρχουν δύο .png εικόνες που δείχνουν το accuracy για δύο αλγορίθμους (LinearSVM, RandomForest) σε σχέση με το components. Σαν $component = 0$, στο γράφημα φαίνεται το accuracy όταν δεν κάνουμε truncation. Αυτό υπάρχει εκεί για να δούμε πόσο αποδοτικό είναι χωρίς truncation. Παρατηρούμε ότι χωρίς truncation είναι αρκετά αποδοτικό και με πειράματα που έγιναν, τα καλύτερα αποτελέσματα τα παίρνουμε με Naive Bayes και χωρίς truncation.

MultinomialNB

Ο CountVectorizer δίνει καλύτερα αποτελέσματα από τον TfidfVectorizer. Παρόλα αυτά όμως αν θέσουμε την παράμετρο $alpha$ του MultinomialNB να είναι 0.001, τότε παίρνουμε παρόμοιο accuracy είτε με CountVectorizer, είτε με TfidfVectorizer. Επίσης φαίνεται να δίνει πολύ καλά αποτελέσματα σαν classifier. Πολλές φορές μάλιστα δίνει καλύτερα αποτελέσματα από όλους τους άλλους classifiers παρόλο ότι είναι ο πιο απλός και κάνει την υπόθεση της ανεξαρτησίας μεταξύ των λέξεων. Επιπλέον, είχαμε καλύτερα αποτελέσματα όταν δεν χρησιμοποιούσαμε το ngram_range=(1, 2) στον vectorizer κάτι που μάλλον μας δείχνει ότι όταν θέλουμε πιο πολλή πληροφοριά παίρνουμε στην πραγματικότητα πιο πολύ θόρυβο.

SVM

Χρησιμοποιήθηκε για δοκιμές η συνάρτηση GridSearchCV όπως φαίνεται και στον κώδικα. Ο τρόπος που λειτουργεί το SVM είναι να μεταφέρει τα δεδομένα σε μεγαλύτερη διάσταση μέχρι να μπορεί να βρει υπερεπίπεδο που να διαχωρίζει τα δεδομένα αποδοτικά.

Linear Kernel Ο TfidfVectorizer δίνει καλύτερα αποτελέσματα από ότι ο CountVectorizer. Επίσης φαίνεται να δουλεύει καλύτερα αυτός ο classifier απο όλους τους άλλους SVM. Έγιναν δοκιμές πάνω στην παράμετρο C, η οποία για μεγάλες τιμές προσπαθεί να ταξινομήσει τα train data όσο καλύτερα γίνεται, αφήνοντας όμως μικρό margin, ενώ με μικρές τιμές αφήνει μεγαλύτερο margin για τα μελλοντικά test data. Στα data μας φαίνεται ότι με την default τιμή που είναι 1, παίρνουμε καλά αποτελέσματα.

RBF Η παράμετρος gamma ορίζει την διακύμανση. Για μεγάλο gamma ένα σημείο επηρεάζει ένα άλλο μόνο αν βρίσκονται κοντά. Ενώ για μικρό gamma, ένα σημείο επηρεάζει ένα άλλο ακόμα και αν βρίσκονται μακριά. Από δοκιμές φαίνεται ότι για $C = 10, gamma = 0.1$ παίρνουμε καλά αποτελέσματα.

RandomForest

Ο αλγόριθμος αυτός ενώ διαισθητικά φαίνεται ότι θα είναι ο καλύτερος, επειδή δημιουργεί πολλά δέντρα απόφασης πάνω στα χαρακτηριστικά, πρακτικά εδώ φαίνεται να δίνει χειρότερα αποτελέσματα από τους Naive Bayes και SVM.

KNN

Ο κώδικας του KNN βρίσκεται στο αρχείο 'knn.py'. Η απόσταση που χρησιμοποιήθηκε είναι η ευκλείδια. Ο αλγόριθμος που χρησιμοποιήθηκε για τον κοντινότερο γείτονα ήταν ο πιο απλός. Δηλαδή αυτός που ελέγχει όλα τα σημεία. Τα αποτελέσματα που πήραμε ήταν καλά, αλλά όχι τόσο όσο του Naive Bayes και του SVM.

My Method

Δύο ήταν οι αλγόριθμοι που χρησιμοποιήθηκαν για την βελτίωση του accuracy. Ο Naive Bayes και ο LinearSVM. Το πιο σημαντικό πράγμα που έγινε σε αυτό το βήμα ήταν να 'καθαρίσουμε' τα κείμενα καλύτερα. Για αρχή βγήκαν όλοι οι παράξενοι χαρακτήρες που δεν ήταν utf8 καθώς και οι αριθμοί, αφού δεν μας δίνουν κάποια σημαντική πληροφορία. Επιπλέον προσθέθηκαν extra λέξεις στις stopWords. Αυτές οι λέξεις βρέθηκαν μετά από αρκετή αναζήτηση και σύγκριση των αποτελεσμάτων κάθε πειράματος, καθώς και από τα wordClouds. Μέσα από τα οποία μπορούμε να βρούμε ποιες λέξεις είναι συχνές σε πολλά κείμενα. Σε επίπεδο stemming, αντικαταστήθηκε το PorterStemmer() που χρησιμοποιήθηκε στην αρχή με το SnowBallStemmer() το οποίο ανέβασε την απόδοση. Επιπλέον η σειρά που ακολουθήθηκε στην προεπεξεργασία ήταν η εξής. Πρώτα αφαιρούμε τα stopWords μαζί με τα νούμερα. Μετά κάνουμε lemmatization που δημιουργεί πάλι πραγματικές λέξεις(την ρίζα κάθε λέξης), οπότε μετά ξανά βγάζουμε τα stopWords. Τέλος κάνουμε stemming για να 'κόψουμε' τις λέξεις. Στον vectorizer τελικά χρησιμοποιήθηκαν οι παράμετροι max_df και min_df, με τιμές που προκύψαν από πειράματα. Επιπλέον ένας άλλος τρόπος για να ανεβάσουμε την απόδοση ήταν να βάλουμε στο υπάρχον trainData set καινούργια κείμενα με λέξεις που υπάρχουν αρκετές φορές για κάθε κατηγορία. Αυτές οι λέξεις βρέθηκαν με την βοήθεια του wordCloud. Βέβαια επειδή δεν προσθέθηκαν πολλά τέτοια κείμενα, η απόδοση έμεινε σχεδόν σταθερή. Όσον αφορά το truncation στο SVM, σε αυτό το στάδιο δεν εφαρμόστηκε, παρόλο που το γράφημα μας δείνχει ότι με συγκεκριμένους components παίρνουμε καλύτερη απόδοση. Ο λόγος ήταν ότι χωρίς truncation η απόδοση ήταν αρκετά κοντά με την καλύτερη του truncation, ενώ στα testData πέρναμε καλύτερα αποτελέσματα χωρίς truncation (τουλάχιστον όσα μπορούσαμε να ελέγξουμε). Ακόμα, επειδή το γράφημα δεν εφαρμόστηκε σε όλα τα δεδομένα (λόγω υπολογιστικής ικανότητας και χρόνου), υπήρχε το ενδεχόμενο με περισσότερα δεδομένα τα ίδια components να μην δίναν την ίδια απόδοση.

Statistics

Statistic Measure	Naive Bayes	Random Forest	SVM	KNN	My Method
Accuracy	0.944	0.83	0.94	0.925	0.962
Precision	0.939	0.835	0.940	0.928	0.962
Recall	0.944	0.799	0.92	0.934	0.951
F-Measure	0.941	0.804	0.931	0.928	0.959