ΜΙΧΑΗΛ ΜΠΙΖΙΜΗΣ AM: **1115201500102**

ΚΩΝΣΤΑΝΤΙΝΤΟΣ ΤΣΙΑΡΑΣ AM: **1115201500165**

ΤΕΧΝΙΚΕΣ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΙΑ 1η

Classification report

Κώδικας

Στον φάκελο Classification περιέχονται:

- Το αρχείο diagnostics.py για εξαγωγή διαγνωστικών στοιχείων σε σχέση με το train_set.csv, η χρήση των οποίων εξηγείται στη συνέχεια.
- Το αρχείο KNN_classifier.py με την δικιά μας υλοποίηση του KNN.
- Τα αρχεία nbayes.py, forest.py, svm.py και knn.py που υλοποιούν τις συναρτήσεις *predict()* και *crossvalidation()* χρησιμοποιώντας το καθένα τον classifier που αναφέρεται στο όνομα του αρχείου (τα πρώτα τρία χρησιμοποιώντας τις αντίστοιχες βιβλιοθήκες του sklearn, το τελευταίο κάνοντας import το δικό μας KNN_classifier.py).
- Το αρχείο svm_gridSearch.py το οποίο, όπως περιγράφεται στο «Παράμετροι του SVM classifier» παρακάτω, υπολογίζει και τυπώνει τις βέλτιστες hyperparameters για τον SVM.
- Το αρχείο components_graph.py το οποίο υπολογίζοντας το accuracy για διάφορες τιμές components στο *TruncatedSVD()* δημιουργεί το γράφημα Components-Accuracy.
- Το αρχείο evaluationMetrics.py το οποίο μέσω 10-fold cross validation υπολογίζει τις ζητούμενες μετρικές (Accuracy, Precision, Recall, F1) για όλες μας τις μεθόδους και δημιουργεί το αρχείο EvaluationMetric_10fold.csv.
- Το αρχείο my_method.py το οποίο περιλαμβάνει την βέλτιστη μέθοδο μας και το οποίο, τρέχοντάς το, κάνει πρόβλεψη για το test_set.csv αποθηκεύοντάς την στο testSet_categories.csv.

Τόσο για τα παραπάνω αρχεία όσο και για το wcloud.py που κατασκευάζει τα wordclouds του ερωτήματος 1, τα datasets θεωρούμε πως υπάρχουν σε έναν φάκελο “./datasets/” στον φάκελο του παραδοτέου.

Μελέτη του train set

Αρχικά, δημιουργήσαμε και τρέξαμε το diagnostics.py, το οποίο συλλέγει τα εξής διαγνωστικά για το train_set.csv:

i. Το πλήθος των άρθρων για κάθε κατηγορία.

ii. Το μέγιστο και το μέσο μήκος του πεδίου 'Content' για όλες τις εγγραφές άρθρων.

Χρησιμοποιήσαμε το πρώτο έτσι ώστε να αποφασίσουμε τι είδους Kfold cross validation να εφαρμόσουμε: απλή KFold ή Stratified. Αν και η Stratified είναι καλύτερη εξ ορισμού, αν οι κατηγορίες ήταν σχετικά ισομοιρασμένες στο train_set.csv και θέλαμε κάτι γρηγορότερο θα μπορούσαμε να θεωρητικά να βασιστούμε στην KFold. Τα αποτελέσματα του i. , όμως, ήταν:

Politics: 2683 Football: 3121 Technology: 1487

Business: 2735 Film: 2240

Συγκεκριμένα, επειδή κάνουμε 10-fold cross validation και έχουμε 12265 συνολικά άρθρα στο train-set.csv κάθε fold θα έχει περίπου 12265/10 = 1226 άρθρα. Επειδή, όμως, τα άρθρα με κατηγορία Technology είναι επικύνδυνα κοντά στο 1226, πράγμα που σημαίνει ότι η πιθανότητα να μην υπάρχουν αρκετά άρθρα Technology σε κάποιο fold είναι σχετικά μεγάλη (σε αντίθεση πχ με αυτά με κατηγορία Football), θεωρήσαμε πιο ασφαλές να τρέξουμε την Stratified KFold, η οποία εγγυάται ότι κάθε fold θα έχει περίπου το ίδιο ποσοστό άρθρων από κάθε κατηγορία με το αρχικό train_set.csv, παρά το έξτρα κόστος της σε χρόνο εκτέλεσης.

Χρησιμοποιήσαμε το δεύτερο για να αποφασίσουμε πόση βαρύτητα να δώσουμε στον τίτλο (εξηγείται παρακάτω).

Προεπεξεργασία των train set και test set

Επειδή η προεπεξεργασία των train και test set δύναται να είναι κοινή για όλες τις μεθόδους εξόρυξης που χρησιμοποιούμε (με εξαίρεση τον *naive bayes* στον οποίον δεν εφαρμόζουμε LSI, καθώς τα features πάνω στα οποία κάνει fit δεν μπορούν να είναι αρνητικά), ό,τι θα εφαρμόζαμε για το «my method» στο ερώτημα «*3. Beat the Benchmark»,* το εφαρμόσαμε για όλες τις μεθόδους. «Βαφτίσαμε», έτσι, my method, αυτήν με το καλύτερο «σκορ» στο 10-fold cross validation.

Συγκεκριμένα, η προεπεξεργασία που κάνουμε πάνω στα train και test set, με την ακόλουθη σειρά, είναι η εξής:

i. Προκειμένου να χρησιμοποιήσουμε αποτελεσματικά την πληροφορία του **τίτλου**, σκεφτήκαμε να προσθέσουμε τον κάθε τίτλο στο text που φυλάμε για κάθε άρθρο έναν παραμετροποιήσιμο αριθμό από φορές (κατά ελάχιστον μία φορά). Αρχικά, προσθέταμε ολόκληρο τον τίτλο ένα fixed amount από φορές, και πειραματικά βλέπαμε πιο fixed amount μας έδινε καλύτερα αποτελέσματα. Αργότερα, σκεφτήκαμε να προσθέτουμε τον τίτλο για ένα άρθρο τόσες φορές ανάλογα με το πόσο μεγάλο είναι το content του. Έτσι, αν ένα άρθρο έχει μικρό content τότε θα προσθέσουμε τον τίτλο αυτού στο συνολικό text για αυτό το άρθρο λιγότερες φορές από ότι θα προσθέταμε τον τίτλο στο text ενός άρθρου που έχει μεγάλο content. Θεωρούμε πως αυτό είναι πιο λογικό σαν κανόνας, αφού έτσι δίνεται η ίδια βαρύτητα στον τίτλο του κάθε άρθρου.

Η βαρύτητα αυτή είναι παραμετροποιήσιμη (η μεταβλητή *mult* στην *add_titles()*), και έτσι δοκιμάσαμε διάφορες τιμές. Η καλύτερη επιλογή φαίνεται να είναι το 0.001 . Από το *diagnostics.py* βρήκαμε ότι κατά μέσο όρο ο αριθμός των λέξεων κάθε πεδίου content είναι περίπου 4000, που σημαίνει ότι κατά μέσο όρο ο τίτλος προστίθεται $\max(1, 0.001 \times 4000) = \max(1, 4) = 4$ φορές, ενώ ο μέγιστος αριθμός των λέξεων αυτών είναι 50000 , που σημαίνει ότι ο μέγιστος αριθμός των φορών που προσθέτουμε τον τίτλο είναι $\max(1, 0.001 \times 50000) = 50$ φορές.

ii. Επίσης, σε αυτό το σημείο κάνουμε αφαίρεση των **stop words** από το text κάθε άρθρου (που τώρα περιέχει και τον τίτλο του), έτσι ώστε λέξεις που υπάρχουν σε όλα τα κείμενα και δεν βοηθούν στην κατηγοριοποίησή τους (πχ: is, that, the, he, …) να μην λαμβάνονται υπόψιν. Αυτές οι λέξεις είναι τα *ENGLISH_STOP_WORDS* από το *sklearn.feature_extraction.text* συν όποια άλλη τέτοια λέξη αντιληφθήκαμε ότι υπάρχει αρκετές φορές μέσα σε αυτά τα κείμενα, όταν φτιάξαμε το *wordcloud* του ερωτήματος 1.

iii. Ύστερα, εφαρμόζουμε ***stemming and lemmatization*** για τα παραπάνω texts, έτσι ώστε παρόμοιες λέξεις (συντακτικά) να θεωρηθούν ως η ίδια λέξη. Αυτή η διαδικασία φάνηκε να παίρνει αρκετό χρόνο, αλλά μας φαίνεται αφελές να μην την εφαρμόσουμε, καθώς μπορεί να βελτιώσει δραματικά την κατηγοριοποίηση οποιασδήποτε μεθόδου.

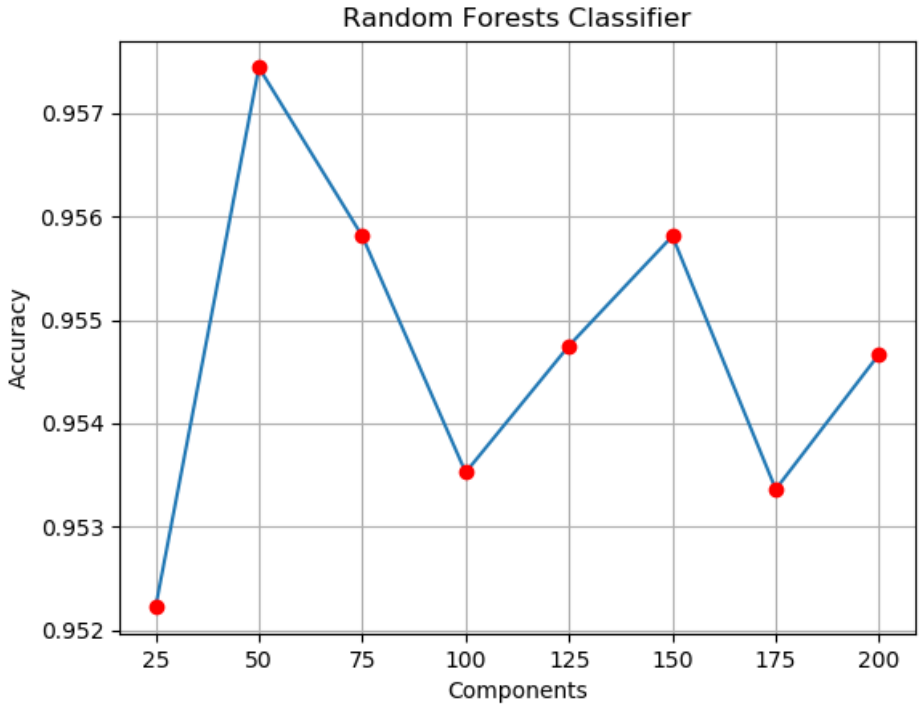
iv. Σε αυτό το σημείο, το text κάθε άρθρου είναι έτοιμο να μετατραπεί σε vector έτσι ώστε να χρησιμοποιηθεί από τις μεθόδους που θα χρησιμοποιήσουμε. Αυτό το κάνουμε χρησιμοποιώντας έναν ***TfidfVectorizer*** και αυτό γιατί φαίνεται να είναι η ευρέως πιο αποδεκτή μέθοδος για vectorization, αφού, εκτός από την συχνότητα εμφάνισης μίας λέξης, ο Tfidf vectorizer λαμβάνει υπόψιν και το πόσες φορές υπάρχει σε όλα τα κείμενα γενικότερα.

Σχετικά με τις παραμέτρους του Tfidf vectorizer, διαβάσαμε στο λινκ: http://scikit-learn.org/stable/modules/decomposition.html#truncated-singular-value-decomposition-and-latent-semantic-analysis ότι είναι προτεινόμενο να χρησιμοποιηθεί η παράμετρος $sublinear\_tf = True$, αν πρόκειται να χρησιμοποιηθεί LSI μετά πάνω σε αυτόν τον vector. Παρ'όλ'αυτά, δοκιμάζοντας το καταλήξαμε σε χειρότερα σκορ στην 10-fold cross validation και έτσι αποφασίσαμε να μην την χρησιμοποιουμε.

v. Τέλος, για όλες τις μεθόδους εκτός από τον *Naive Bayes* όπου τα features δεν μπορούν να είναι αρνητικά, εφαρμόζουμε **LSI**, προκειμένου να «συμπυκνώσουμε» τα γενικώς sparse vectors (θα έχουν πολλά μηδενικά), που δημιουργούνται από την παραπάνω διαδικασία, μιας και το vector περιέχει πληροφορία (έναν αριθμό) για κάθε λέξη που εμφανίζεται σε όλα τα κείμενα κάθε άρθρου, ενώ ένα μόνο άρθρο, λογικά, θα περιέχει ένα σχετικά περιορισμένο ποσοστό όλων αυτών.

Η επιλογή αριθμού των components για LSI έγινε ξεχωριστά για κάθε αλγόριθμο, ως εξής:

- Για τον Random Forests classifier, με σταθερό *random_state=13*, πήραμε το παρακάτω γράφημα Components-Accuracy, σύμφωνα με το οποίο διαλέξαμε *n_components=50:*



- Για τον SVM δοκιμάσαμε διάφορα components και πειραματικά διαπιστώσαμε πως, μέχρι τα 200, μεγαλύτερο πλήθος components είχε ως αποτέλεσμα υψηλότερα metric scores ενώ από εκεί και μετά άρχισαν να χειροτερεύουν. Για το λόγο αυτό διαλέξαμε *n_components=200*.

- Για τον KNN, καθώς ο χρόνος εκτέλεσης του ήταν απαγορευτικός για μεγάλο πλήθος components περιοριστήκαμε σε *n_components=5*.

Παράμετροι του SVM classifier

Χρησιμοποιώντας την παραπάνω εκτιμώμενη βέλτιστη τιμή για τα components ($n = 200$), τρέξαμε το svm_gridsearch.py για εύρεση των καλύτερων hyperparameters, από το οποίο προέκυψαν:

- $C = 10$
- $gamma = 0.001$
- $kernel = 'linear'$

Cross validation metrics και τελική επιλογή

Statistic Measure	Naive Bayes	Random Forest	SVM	KNN	My Method
Accuracy	0.935020	0.957442	0.968040	0.935673	0.968040
Precision	0.941770	0.955021	0.965347	0.929548	0.965347
Recall	0.910027	0.953451	0.966318	0.926677	0.966318
F-Measure	0.918984	0.954094	0.965726	0.927909	0.965726

Έχοντας κάνει την ίδια προεπεξεργασία στα κείμενα με όλους τους classifiers, ως “My Method” επιλέξαμε τη μέθοδο που μας έδινε τα καλύτερα metric scores, δηλαδή τον SVM.

Οι προβλέψεις κατηγοριών των άρθρων που περιέχονται στο test_set.csv με τη χρήση της My Method βρίσκονται στο αρχείο testSet_categories.csv.