Τεχνικές Εξόρυξης Δεδομένων
Εαρινό Εξάμηνο 2017-2018

1η Άσκηση

Γιώργος Μανδηλαράς Α.Μ.: 1115201200097

Θοδωρής Μανδηλαράς Α.Μ.:1115201200098

1. WordCloud

Τα wordclouds δημιουργήθηκαν με την χρήση της βιβλιοθήκης wordcloud. Για να παραχθούν τα wordcloud δημιουργήσαμε μία μεταβλητή για την κάθε κατηγορία, την οποία γεμίσαμε με τα άρθρα της κάθε κατηγορίας. Στην συνέχεια εφαρμόσαμε την συνάρτηση create_WordCloud για καθεμιά από αυτές τις μεταβλητές, και έτσι παράγαμε τα wordclouds για την κάθε κατηγορία.

Τα wordclouds παράγονται από το αρχείο **WordCloud_producer.py**, αποθηκεύονται στον φάκελο WordClouds, και είναι τα εξής:

Business

Films

Football

Politics

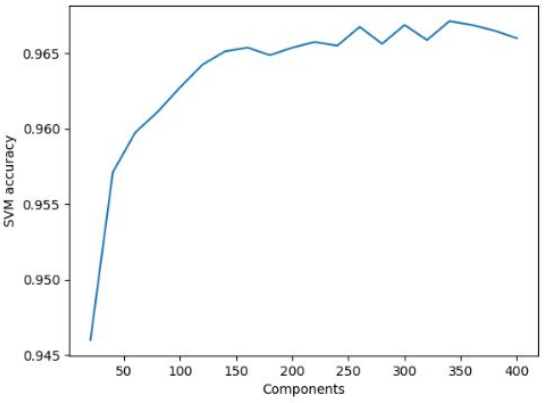
Technology

2. Υλοποίηση Κατηγοριοποίησης (Classification)

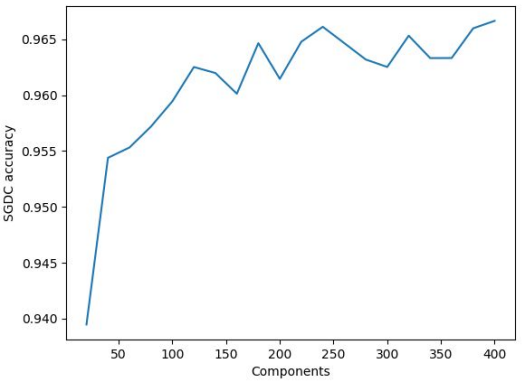
Η υλοποίηση αυτού του ερωτήματος έγινε στο αρχείο **classifiers.py**. Η διαδικασία είναι η εξής:

Αρχικά αποθηκεύσαμε το περιεχόμενο στηλών Content και Title του αρχείου train_set.csv σε μία μεταβλητή X, και το περιεχόμενο των στηλών Category σε μια μεταβλητή y αφού πρώτα εφαρμόσαμε label encoder. Επειδή θέλαμε να δώσουμε μεγαλύτερη έμφαση στις λέξης του τίτλου, προσθέσαμε των τίτλο 10 φορές στην μεταβλητή X, ωστόσο δοκιμάσαμε και με περισσότερες φορές αλλά οδήγησε στο να χειροτερέψουν τα τελικά αποτελέσματα. Στην συνέχεια κάναμε μια τυπική προ-επεξεργασία μετατρέποντας τα κεφαλαία σε μικρά και αφαιρώντας μερικά σύμβολα. Επίσης εφαρμόσαμε stemming, δηλαδή κόψαμε τις καταλήξεις των λέξεων ως την ρίζα τους (δοκιμάσαμε και lemmatization αλλά δεν παρατηρήσαμε κάποια βελτίωση στα αποτελέσματα).

Έπειτα μετατρέψαμε τα κείμενα σε διαστήματα λέξεων με την χρήση του tfidfVectorizer, μιας και έδινε καλύτερα τελικά αποτελέσματα από τον CountVectorizer. Ωστόσο σκεφτήκαμε να κάνουμε μια επιπλέον επεξεργασία η οποία ήταν να διπλασιάσουμε τις τιμές των πιο συχνά εμφανιζόμενων λέξεων, δηλαδή όποια λέξη είχε στο διάνυσμα τιμή μεγαλύτερη από 0.5 να την διπλασιάζαμε. Με αυτήν την ενέργεια θέλαμε να δώσουμε μεγαλύτερη έμφαση στις πιο συχνά εμφανιζόμενες λέξεις, ωστόσο δεν αύξησε σημαντικά την απόδοση των περισσότερων αλγορίθμων, και μείωσε την απόδοση του Naive Bayes αλλά αύξησε κατά λίγο την απόδοση του KNN.



Στην συνέχεια εφαρμόσαμε Latent Semantic Indexing με την χρήση της συνάρτησης TruncatedSVD, ωστόσο για να βρούμε τον βέλτιστο αριθμό components χρειάστηκε να δημιουργήσουμε το αρχείο component_graph_producer.py το οποίο τρέχει τον αλγόριθμο svm με διάφορες τιμές component και παράγει ένα γράφημα accuracy - components, έτσι βρήκαμε πως ο βέλτιστος αριθμός component για τον SVM είναι 340,



ωστόσο επειδή ο αριθμός ήταν πολύ μεγάλος χρησιμοποιήσαμε ένα τοπικό μέγιστο , το 160 (πιστεύαμε πως ο μεγάλος αριθμός components προκαλούσε το MemoryError - αναφέρεται πιο κάτω). Παράδειγμα του γραφήματος:

Έπειτα κάναμε 10-fold Cross Validation με την χρήση της συνάρτησης StratifiedKFold(n_splits=10) και εφαρμόσαμε τους αλγορίθμους Naive Bayes, Random Forest, SVM, KNN και την μέθοδο της επιλογής μας την SGDClassifier.

Στον SVM χρησιμοποιήσαμε την εντολή GridSearchCV για να βρούμε της βέλτιστες τιμές για τις παραμέτρους kernel, C και gamma, ωστόσο αυτό που έκανε ήταν να δοκιμάζει όλες τις τιμές που του δίναμε και να επιλέξει τις καλύτερες με αποτέλεσμα να καθυστερεί αρκετά σημαντικά την ολοκλήρωση του προγράμματος. Για αυτό εμείς την χρησιμοποιήσαμε για να βρούμε τις καλύτερες τιμές, οι οποίες είναι kernel = 'rbf', C = 1 και gamma = 1, και τις εκχωρήσαμε κατευθείαν στον SVM.

Για τον Naive Bayes χρησιμοποιήσαμε την συνάρτηση MultinomialNB(), όμως επειδή δεν δεχόταν αρνητικούς αριθμούς, οι οποίοι είχαν παραχθεί από το lsi, εφαρμόσαμε MinMaxScaler σε διάστημα 1 - 100. Ωστόσο αυτό μείωσε αρκετά την απόδοση του αλγορίθμου από περίπου 95% στο 90%.

Για την υλοποίηση του knn δημιουργήσαμε το KNN_Classifier.py στο οποίο αρχικά βρίσκουμε τις ευκλείδειες αποστάσεις του διανύσματος που θέλουμε να κατηγοριοποιήσουμε με όλα τα άλλα διανύσματα τα οποία έχουμε για την εκπαίδευση του αλγορίθμου, και τις αποθηκεύουμε σε μια λίστα. Στην συνέχεια ταξινομούμε την λίστα και διαλέγουμε τα k πρώτα. Έπειτα εφαρμόζουμε κατηγοριοποίηση με βάση majority voting, δηλαδή το κατηγοριοποιούμε στην κατηγορία την οποία ανήκουν τα περισσότερα από τα k πρώτα διανύσματα, που βρίσκονται πιο κοντά σε αυτό. Αυτό το επιτυγχάνουμε με την χρήση της εντολής Counter από την βιβλιοθήκη collections.

Για να υπολογίσουμε τις μετρικές που μας ζητήθηκαν χρησιμοποιούμε την βιβλιοθήκη metrics. Ωστόσο επειδή κάνουμε 10-fold cross validation, υπολογίζουμε τα αποτελέσματα σε κάθε επανάληψη, και στο τέλος παίρνουμε τον μέσον όρο. Στην συνέχεια κατασκευάζουμε το αρχείο Produced_Files/EvaluationMetric_10fold.csv με την βοήθεια της βιβλιοθήκης pandas.

EvaluationMetric_10fold.csv

Statistic Measure	Naive Bayes	Random Forest	SVM	KNN	Stochastic Gradient Descent
Accuracy	0.9031005354	0.9468951289	0.9653973513	0.9296979885	0.9628976478
Precision	0.9031005354	0.9468951289	0.9653973513	0.9296979885	0.9628976478
Recall	0.9031005354	0.9468951289	0.9653973513	0.9296979885	0.9628976478
F-Measure	0.9031005354	0.9468951289	0.9653973513	0.9296979885	0.9628976478

3. Beat the Benchmark

Η μέθοδος η οποία έβγαλε τα καλύτερα αποτελέσματα στο 10-fold Cross Validation ήταν ο SVM, ο οποίος κατά ελάχιστο ξεπερνούσε τον ταξηνομιτή της επιλογής μας, τον SGDClassifier. Για την επεξεργασία των δεδομένων κάναμε ότι αναφέραμε στην προηγούμενη ενότητα, δηλαδή καθαρισμό του κειμένου από κεφαλαία και σύμβολα, stemming, tfidfVectorizer, διπλασιασμό των ακραίων τιμών και lsi με τον αριθμό των components που παρουσιάζει μέγιστο accuracy (components = 160).

Ωστόσο αντιμετωπίσαμε το εξής ΠΡΌΒΛΗΜΑ. Τις περισσότερες φορές που το τρέχαμε με ολόκληρο το tain_set έσκαγε MemoryError στο lsi ασχέτως τον αριθμό των components που δίναμε. Αυτό έκανε πολύ δύσκολο την ολοκλήρωση του ερωτήματος μιας και στους υπολογιστές μας έπαιρναν πάρα πολύ χρόνο για να εκτελέσουν το πρόγραμμα, και με τις περισσότερες φορές να σκάει.