ΤΕΧΝΙΚΕΣ ΕΞΟΡΥΞΗΣ ΔΕΔΟΜΕΝΩΝ

Εργασία 1

Σταματόπουλος Βασίλειος – 1115201400188
Κουκάκης Κωνσταντίνος - 1115201400289

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

WORDCLOUD

Στο πρώτο μέρος της εργασίας, υλοποιήσαμε ένα wordcloud για τις λέξεις που περιέχονται στα άρθρα των διάφορων κατηγοριών στα δεδομένα. Σκοπός είναι να χωρίσουμε τα δεδομένα σε έναν πίνακα από features, όπου κάθε στήλη είναι ένα ξεχωριστό χαρακτηριστικό με το 5 εξ αυτών να είναι η κατηγορία η οποία μας ενδιαφέρει. Έτσι, για κάθε άρθρο αναλόγως την κατηγορία του, θα τοποθετήσουμε το κείμενό του σε έναν πίνακα ο οποίος θα περιέχει όλα τα κείμενα της κατηγορίας αυτής. Τέλος, θα καλέσουμε την συνάρτηση δημιουργίας Wordcloud από το wordcloud module, για να δημιουργήσουμε τα 4 wordclouds.

Καθώς πολλές λέξεις χρησιμοποι

Το αρχείο python με το οποίο δημιουργούνται τα wordclouds, είναι το wc.py και σε αυτό υλοποιήσαμε με κώδικα τα παραπάνω. Χρησιμοποιούμε τους πίνακες text_p, text_ft, text_f, text_t, text_b, για το content της κάθε κατηγορίας και έπειτα δημιουργούμε τα wordclouds μέσω του Wordcloud() function. Χρησιμοποιούμε τα δικά μας stopwords σε συνδυασμό με τα ENGLISH_STOP_WORDS, και σώζουμε το κάθε wordcloud σε ένα αρχείο png στο φάκελο WordClouds.

CLASSIFIERS

Υλοποιήθηκαν οι παρακάτω αλγόριθμοι

1) *Random Forest*
2) *Naïve Bayes*
3) *Support Vector Machine*
4) *K-Nearest Neighbor*

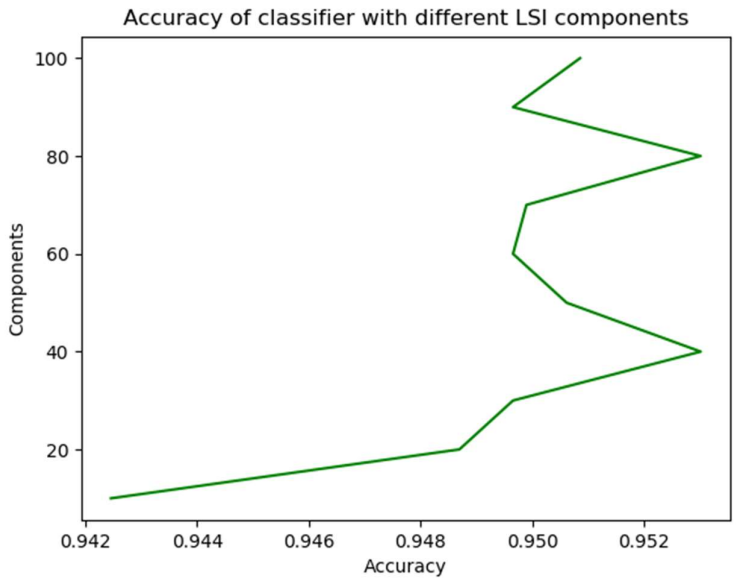
Οι αλγόριθμοι αυτοί υλοποιήθηκαν σε functions και καθένα από αυτά βρίσκεται στα αντίστοιχα αρχεία: RF_classifier.py, NB_classifier.py, SVM_classifier.py, KNN_classifier.py. Στο αρχείο main.py, κάνουμε import τα προηγούμενα functions και έπειτα από προεπεξεργασία στα δεδομένα, τα καλούμε.

Η προεπεξεργασία που γίνεται είναι η διανυσματοποίηση των χαρακτηριστικών, η δημιουργία των εξαρτημένων μεταβλητών y και στους 3 από τους 4 ταξινομητές εφαρμόζουμε Latent Semantic

Indexing. Για τις διαδικασίες αυτές χρησιμοποιούμε τα modules TfidVectorizer(), LabelEncoder() και TruncatedSVD. Η πρώτη συνάρτηση που χρησιμοποιείται για τη **διανυσματοποίηση**, έχει το θετικό πως δίνει βάρη στις λέξεις ανάλογα με τη συχνότητα εμφάνισής τους. Έτσι, λέξεις οι οποίες χρησιμοποιούνται πολλές φορές στο σύνολο δεδομένων και περιέχουν μικρή πληροφορία(π.χ άρθρα), δεν θα έχουν μεγάλη επιρροή στην ταξινόμηση. Στη διαδικασία αυτή βοηθάει και η συλλογή με τα stopwords.

Η συνάρτηση TruncatedSVD(), πραγματοποιεί **μείωση διαστάσεων** (dimensionality reduction) σε διανυσματοποιημένα δεδομένα. Συγκρίνει τις παρόμοιες λέξεις και δίνει σε αυτές τιμές κοντά στο 1, ενώ στις διαφορετικές τιμές κοντά στο 0. Γενικότερα, το latent semantic analysis πρεσβεύει πως παρόμοιες λέξεις θα υπάρχουν σε παρόμοια άρθρα-documents. Η παράμετρος **n_components,** ορίζει των αριθμό των μειωμένων διαστάσεων. Τα χαρακτηριστικά που διανυσματοποιούνται είναι ο **Τίτλος** και το **Περιεχόμενο**.

Για την εύρεση του κατάλληλου αριθμού components, χρησιμοποιήθηκε μια επαναληπτική διαδικασία η οποία εφαρμόζει τον ταξινομητή για διαφορετικά components σε κάθε επανάληψη. Για αυτή τη διαδικασία χρησιμοποιήθηκαν (10,20,30,40,50,60,70,80,90,100) components και προέκυψε το παρακάτω διάγραμμα, στο οποίο φαίνεται πως η καλύτερη επιλογή components βάση της ακρίβειας του ταξινομητή, είναι το 40.



Οι συναρτήσεις επιστρέφουν τα μετρικά (accuracy, precision, recall, f1) των αλγορίθμων, τα οποία εισάγονται σε 4 πίνακες αντίστοιχα και βάση των πινάκων αυτών δημιουργείται το αρχείο EvaluationMetric_10fold.csv.

Naïve Bayes

Ο αλγόριθμος αυτός δεν επεξεργάζεται δεδομένα που έχουν υποστεί μείωση διάστασης, καθώς δεν μπορεί να επεξεργαστεί αρνητικά στοιχεία. Παρ' όλα αυτά όπως και οι υπόλοιπες υλοποιήσεις που θα ακολουθήσουν, χρησιμοποιεί 10 fold cross validation, χωρίζοντας το σύνολο X σε 9 train samples και 1 test sample για 10 επαναλήψεις. Έτσι, κάθε sample χρησιμοποιείται ως test set, μειώνοντας τον κίνδυνο υπερπροσαρμογής. Τέλος, επιστρέφει το σύνολο των μετρικών των 10 επαναλήψεων δια του 10 για να δώσει τις μέσες αποδόσεις του αλγορίθμου.

Random Forest

Ο αλγόριθμος αυτός σε αντίθεση με τον προηγούμενο χρησιμοποιεί το TruncatedSVD(). Όλα τα υπόλοιπα παραμένουν ίδια, με μόνη διαφορά το imported module και τη συνάρτηση που χρησιμοποιείται.

Support Vector Machine

Η διαφορά σε αυτή την υλοποίηση του αλγορίθμου, ήταν πως χρησιμοποιήθηκε η συνάρτηση βιβλιοθήκης GridSearchCV, για να βρεθούν οι βέλτιστοι παράμετροι. Η συνάρτηση αυτή τρέχει των αλγόριθμο με όλες τους διαφορετικούς συνδυασμούς παραμέτρων που του θέτουμε και με το best_params_ επιστρέφει το σύνολο των καλύτερων. Στη περίπτωση αυτή, αυτές ήταν: το rbf kernel, c = 10000, gamma = 0.01.

K-Nearest Neighbors

Για τον αλγόριθμο αυτό δε χρησιμοποιήθηκε κάποια έτοιμη βιβλιοθήκη, αλλά υλοποιήθηκε από εμάς στα πλαίσια της εργασίας.

Καθώς ο αλγόριθμος αυτός είναι μη παραμετρικός, δεν δημιουργεί κάποιο πιθανοτικό μοντέλο, αλλά σώζει στη μνήμη τα δεδομένα εκπαίδευσης και ελέγχει βάση αυτών τα παραδείγματα εξέτασης. Η υλοποίηση έγινε στο KNN_classifier.py.

Αρχικά, μέσα στην επανάληψη **for i in range(len(X_test)),** υπολογίζεται η απόσταση του καινούριου δεδομένου με όλα τα υπόλοιπα, και προστίθεται στον πίνακα distances. Έπειτα, ταξινομείται ο πίνακας αυτός και παίρνουμε τους k κοντινότερους γείτονες. Για το k, συνηθίζεται να χρησιμοποιούνται πολλαπλάσια του 7.

Αποτελέσματα

Τα αποτελέσματα των μετρικών που δίνουν οι αλγόριθμοι δίνονται στον παρακάτω πίνακα

Measures	**Naïve Bayes**	**Random Forest**	**S.V.M**	**K.N.N**
Accuracy	0.9437	0.9554	0.9621	0.9578
Precision	0.9481	0.9526	0.9591	0.9547
Recall	0.9240	0.9508	0.9597	0.9551
F1	0.9319	0.9515	0.9593	0.9548

BEATING THE BENCHMARK

Για το ερώτημα αυτό σχεδιάσαμε ένα Multilayer Perceptron για ταξινόμηση. Ο λόγος που επιλέξαμε αυτή την υλοποίηση, είναι επειδή θέλαμε να δούμε τι απόδοση θα είχε αυτή η μέθοδος στην εργασία αυτή. Γενικότερα, τα νευρωνικά δίκτυα είναι μια αρκετά ενδιαφέρουσα μέθοδος μηχανικής μάθησης και στην εργασία αυτή δώσανε πολύ καλά αποτελέσματα. Πιο συγκεκριμένα, στα άγνωστα δεδομένα του test_set, πήραμε accuracy=0.96521. Ο αλγόριθμος βρίσκεται στο αρχείο my_model.py.

Αρχικά, βγάλαμε τα άχρηστα features από το dataset, και όπως στους προηγούμενους αλγορίθμους δώσαμε labels στις κατηγορίες. Στην προεπεξεργασία, πραγματοποιήσαμε διανυσματοποίηση, lsa αλλά και Standardization, καθώς τα MLPs είναι ευαίσθητα σε Feature Scaling. Επιπλέον, λάβαμε υπόψιν και τον τίτλο στην διανυσματοποίηση.

Με την κανονικοποίηση, μορφοποιούμε τα δεδομένα ώστε να έχουν mean = 0 και μεταβλητότητα = 1. Παρότι εφαρμόζουμε την μεταβολή και στο train αλλά και στο test set, κάνουμε fit μονάχα στο πρώτο.

Οι επιλεγμένες παράμετροι, βρέθηκαν εμπειρικά και είναι οι παρακάτω:

Hidden layers = 3 με 45 κόμβους το καθένα

Maximum Iterations = 500

Alpha = 0.0001

Activation function = Relu

Solver = Stochastic Gradient Descent

Learning Rate = 0.01, adaptive

Tolerability =0.00000000001

Ξεκινήσαμε με 1 κρυφό επίπεδο των 100 κόμβων, το οποίο παρότι έδινε σχετικά καλά αποτελέσματα αργούσε πολύ. Έπειτα, χρησιμοποιήσαμε 2 επίπεδα των 50 κόμβων που δίναν πολύ καλά αποτελέσματα και μετά δοκιμάσαμε δύο επίπεδα των 20, 30 και 40. Από αυτά, τα καλύτερα αποτελέσματα τα έδινε αυτό με το 40. Φτιάχνοντας 3 επίπεδα των 40 και των 50 είδαμε βελτίωση από πριν αλλά μικρή διαφορά μεταξύ των δύο. Έτσι, καταλήξαμε στην επιλογή των τριών επιπέδων με 45 κόμβους το καθένα.

Για την εκτίμηση χρησιμοποιήσαμε SGD, λόγω της μεγάλης ευχέρειας που προσφέρει. Με τον εκτιμητή αυτόν, μπορούμε να ελέγχουμε και την συνάρτηση ενεργοποίησης αλλά και το learning rate. Η στοχαστικότητα του αλγορίθμου, σε συνδυασμό με την προσαρμοστικότητα του δείκτη μάθησης, πιστεύουμε πως ευθύνεται για την απόδοση του αλγορίθμου μας.

Η απόδοση του αλγορίθμου είναι βέβαιο πως μπορεί να βελτιστοποιηθεί. Όμως, καθώς ξεπεράσαμε το benchmark του 0.96 θεωρήσαμε πως πλέον οποιαδήποτε αλλαγή θα έδινε πολύ μικρό όριο βελτίωσης.

RUN THE PROGRAM

python wc.py ->wordcloud

python main.py -> classifiers

python my_model.py -> Multilayer Perceptron