Τεχνικές Εξόρυξης Δεδομένων

Άσκηση 1

Τσεμσελής Δημήτρης – 1115201400208
Φλωράκης Απόστολος – 1115201400217

Παραδοτέα αρχεία με κώδικα:

- **main.py**: Το αρχείο αυτό περιέχει την προεπεξεργασία των δεδομένων και την κλήση των διάφορων ταξινομητών είτε για cross validation είτε για predict του test dataset, με αποσχολιασμό των κατάλληλων γραμμών.
- **cross_validation.py**: Το αρχείο αυτό περιέχει τις συναρτήσεις για cross validation των υλοποιημένων ταξινομητών, του KNN καθώς και του δικού μας ταξινομητή.
- **KNN_classifier.py**: Το αρχείο αυτό περιέχει την υλοποίηση του KNN ταξινομητή.
- **beat_the_bench.py**: Το αρχείο αυτό περιέχει τη συνάρτηση που υλοποιεί το δικό μας ταξινομητή.
- **classifiers_and_stemming.py**: Το περιεχόμενο είναι παρόμοιο με αυτό του main.py, με τη διαφορά ότι γίνεται και χρήση stemming στο pipeline.
- **wordcloud_creator.py**: Το αρχείο αυτό περιέχει τον κώδικα για τη δημιουργία των εικόνων από το Word Cloud.

1.Word Cloud

Για τη σωστή εκτέλεση του κώδικα του αρχείο wordcloud_creator.py χρειάζεται στον ίδιο φάκελο να υπάρχει και η εικόνα με όνομα “eikona.jpg”, η οποία αποτελεί το περίγραμμα των Word Cloud.

2. Υλοποίηση κατηγοριοποίησης (Classification)

Κατά το στάδιο της προ επεξεργασίας των δεδομένων εφαρμόσαμε διάφορες μεθόδους και πειραματισμούς, ώστε να μπορέσουμε να εξάγουμε από τα κείμενα χαρακτηριστικά (features), τα οποία θα βοηθήσουν στην μείωση του σφάλματος κατά τη διαδικασία της κατηγοριοποίησης (classisfication) των κειμένων.

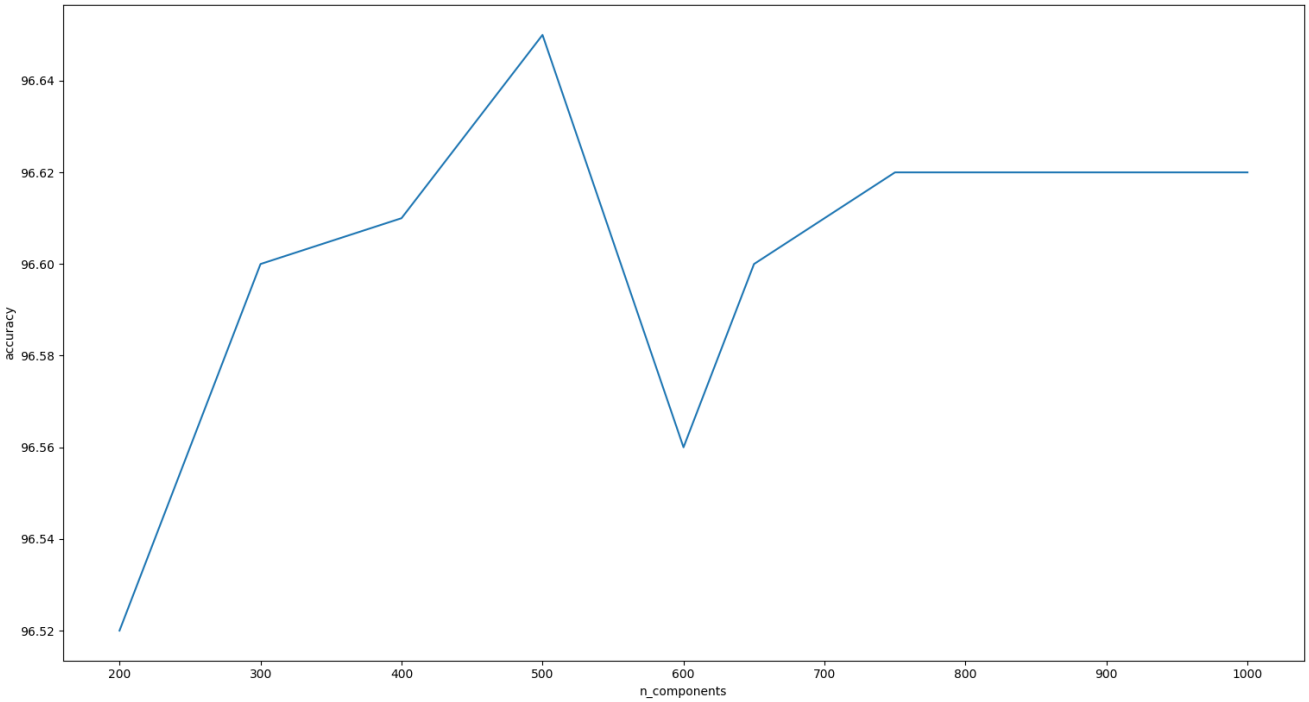
Vectorization:

Απαραίτητη για την ταξινόμηση των κειμένων είναι η εξαγωγή ενός διανύσματος για το κάθε ένα από αυτά (Vectorization). Προτιμήθηκε ο tf-id vectorizer έναντι του count-vectorizer καθώς σε αντίθεση με τον δεύτερο ο οποίος απλά μετράει τις εμφανίσεις κάθε λέξης στο κείμενο και αναθέτει την τιμή στο αντίστοιχο index του διανύσματος, ο tfid vectorizer υπολογίζει τη συχνότητα εμφάνισης κάθε λέξης λαμβάνοντας υπόψιν το συνολικό μέγεθος του αρχείου, παράγοντας έτσι ένα πιο ακριβές και αντιπροσωπευτικό διάνυσμα.

LSI:

Χρησιμοποιείται η τεχνική “Latent Semantic Indexing” (LSI). Συγκεκριμένα μέσω του μετασχηματισμού TruncatedSVD επιτυγχάνεται γραμμική μείωση των διαστάσεων των διανυσμάτων που θα χρησιμοποιήσουμε ως χαρακτηριστικά. Ο μετασχηματισμός αυτός εφαρμόζει μια παραλλαγή της αποσύνθεσης μοναδιαίας τιμής (Singular Value Decomposition – SVD) υπολογίζοντας μόνο τις n_components μεγαλύτερες τιμές όπου n_components μεταβλητή παράμετρος. Όταν ο μετασχηματισμός αυτός εφαρμόζεται σε πίνακες μορφής term-document matrices (όπως επιστρέφονται από τον CountVectorizer ή τον TfidfVectorizer), είναι γνωστός ως λανθάνουσα σημασιολογική ανάλυση (LSA), διότι μετατρέπει τέτοιους πίνακες σε "σημασιολογικό" χώρο χαμηλής διαστάσεων. Συγκεκριμένα, το LSA είναι γνωστό ότι καταπολεμά τα αποτελέσματα της συνωνυμίας (πολλαπλές ερμηνείες ανά λέξη), που προκαλεί αραιά διανύσματα με αποτέλεσμα μετρικές όπως η ομοιότητα συνημίτονων εφαρμοζόμενες στα διανύσματα αυτά να χάνουν την αποτελεσματικότητα τους.

Αριθμός n_components:



Από το γράφημα παρατηρούμε πως το καλύτερο accuracy (96.64%) επιτυγχάνεται για αριθμό n_components = 500. Να σημειωθεί πως η ακρίβεια αυτή προέκυψε με χρήση max_features = 7000, ενώ πειραματιστήκαμε και με διάφορες άλλες τιμές για την παράμετρο αυτή που δεν έδωσαν όμως καλύτερα αποτελέσματα.
Παρατηρώντας το γράφημα συμπεραίνουμε ότι όσο αυξάνεται ο αριθμός των components από 100 έως και 500, υπάρχει αύξηση της ακρίβειας των προβλέψεών μας.
Αυτό μας δείχνει πως αν για παράδειγμα επιλέξουμε 100 μόνο components, μειώνουμε σημαντικά το χρόνο εκτέλεσης του προγράμματος αλλά μειώνουμε επίσης και την ακρίβεια των προβλέψεών μας, καθώς μικραίνοντας
κατά πολύ το μέγεθος των διανυσμάτων, κρατάμε πολύ λίγα από τα χαρακτηριστικά των προτάσεών μας.
Βλέπουμε επίσης πως για 750 έως 1000 components τα αποτελέσματα είναι αρκετά καλά. Παρόλα αυτά, για n_components = 1000 ο χρόνος εκτέλεσης του προγράμματος αυξάνεται αρκετά.

Stop Words:

Από τα κείμενα επιλέξαμε να αφαιρούμε τις λέξεις οι οποίες εμφανίζονται συχνά στην Αγγλική γλώσσα και δεν αποτελούν χαρακτηριστικό και ένδειξη για το περιεχόμενο του κειμένου. Τέτοιες λέξεις είναι άρθρα, σύνδεσμοι αντωνυμίες κλπ αλλά και λέξεις που παρατηρήσαμε στο ερώτημα κατασκευής του wordcoud για κάθε

κατηγορία άρθρου, πως εμφανίζονται πολύ συχνά σε όλους τους τύπους κειμένου ανεξαρτήτου της κατηγορίας τους (πχ: "government", "people", "time" ...). Η προσπάθεια αυτή, δηλαδή ο εμπλουτισμός των ήδη υπάρχοντων STOP_WORDS, φάνηκε πως δεν ήταν ιδιαίτερα χρήσιμη, καθώς παρατηρήθηκε μικρή μείωση της ακρίβειας.

Χρήση Pipeline:

Για την βελτιστοποίησης της διαδικασίας εκμάθησης των αλγορίθμων και της πρόβλεψης των κατηγοριών, χρησιμοποιήσαμε Pipeline. Με τον τρόπο αυτόν τα βήματα Vectorization, Transformation, Truncatedsvd, “πακετάρονται” και εκτελούνται με ορισμένη σειρά στο pipeline με μοναδικό στοιχείο που διαφοροποιεί κάθε μια κλήση της συνάρτησης το είδος του ταξινομητή που δίνεται σαν τελευταίο όρισμα της.

Αξιοποίηση Τίτλου:

Για την αποτελεσματική αξιοποίηση του τίτλου εφαρμόσαμε διάφορες τεχνικές.
Αρχικά, προσπαθήσαμε να εισάγουμε τους τίτλους στο κυρίως κείμενο, προσθέτοντάς τον αντίστοιχο τίτλο στο τέλος του αντίστοιχου κειμένου. Με τον τρόπο αυτό, θέλαμε να δώσουμε μεγαλύτερη σημασία στις λέξεις που εμφανίζονται στον τίτλο, αυξάνοντας την εμφάνισή τους στο κείμενο. Για τη μέθοδο αυτή, δοκιμάσαμε να αντιγράψουμε τους τίτλους από 1 έως 10 φορές στα κυρίως κείμενα, παρόλα αυτά όσο ο αριθμός των αντιγραφών μεγάλωνε, τόσο το accuracy έπεφτε.

Στη συνέχεια, εξάγοντας τις πιο συχνά εμφανιζόμενες λέξεις από τους τίτλους της κάθε κατηγορίας με τη χρήση του WordCloud, προσπαθήσαμε δεδομένης της πρόβλεψης ενός ταξινομητή βασισμένου στο κυρίως κείμενο, να διορθώσουμε τυχόν λάθη βασισμένοι στον τίτλο. Συγκεκριμένα, εάν ο τίτλος του κειμένου περιείχε 2 ή παραπάνω λέξεις που ανήκαν σε μία κατηγορία (πχ: για την Football οι λέξεις “Liverpool” και “World Cup”) τότε αλλάζαμε την πρόβλεψη του κειμένου και το κατατάσσαμε στην κατηγορία αυτή. Η μέθοδος αυτή δεν απέδωσε καλά αποτελέσματα, αφού το accuracy είτε παρέμενε ίδιο με αυτό χωρίς τίτλο είτε έπεφτε έως 4-5%.

Τελικά, η τεχνική που φάνηκε ιδιαίτερα χρήσιμη για την αξιοποίηση του τίτλου είναι η εξής. Αρχικά, αφού ο ταξινομητής εκπαιδευτεί με τα δεδομένα των κυρίως κειμένων, χρησιμοποιείται η συνάρτηση **predict_proba,** η οποία επιστρέφει τις πιθανότητες (μία για κάθε κατηγορία) το εκάστοτε κείμενο προς ταξινόμηση να ανήκει στις κατηγορίες. Έπειτα, γίνεται η ίδια διαδικασία για τον τίτλο, δηλαδή ο ταξινομητής εκπαιδεύεται με τα δεδομένα των τίτλων και παράγει τις πιθανότητες των κειμένων να ανήκουν σε μία από τις κατηγορίες. Τα δύο αυτά διανύσματα αθροίζονται μεταξύ τους και η τελική πρόβλεψη είναι η κατηγορία στην οποία

αντιστοιχεί το μεγαλύτερο άθροισμα πιθανοτήτων. Με τον τρόπο αυτό, καταφέραμε να αυξήσουμε το accuracy κατά 1.5-2%.

Υλοποίηση KNN:

Όσον αφορά τη δική μας υλοποίηση του KNN έχει οριστεί συνάρτηση KNN η οποία ταξινομεί τα κείμενα βάσει των 7 πλησιέστερων γειτόνων τους. Δεν χρησιμοποιούμε pipeline οπότε αρχικά πραγματοποιούμε vectorization, transformation και εφαρμόζουμε truncated_svd για τα train και predict set. Στη συνεχεία υπολογίζουμε για κάθε ένα στοιχείο του predict set τις 7 κοντινότερες αποστάσεις, μέσω της συνάρτησης calculate_distances όπως προκύπτουν μετά από ταξινόμηση κατά αύξουσα σειρά όλων των αποστάσεων. Αποφασίζουμε εν τέλει που θα κατατάξουμε το στοιχείο με κριτήριο majority voting δηλαδή βάσει της κατηγορίας στην οποία ανήκει η πλειοψηφία των 7 αυτών κοντινότερων επιλογών. Συγκεκριμένα πρώτα υπολογίζονται και τοποθετούνται σε ένα Dictionary οι εμφανίσεις κάθε κατηγορίας στα 7 αποτελέσματα που επέστρεψε η συνάρτηση calculate_distances, έπειτα η δομή ταξινομείται κατά φθίνουσα σειρά και τέλος επιλέγουμε την πρώτη θέση, την κατηγορία δηλαδή που αποτελεί την πιο συχνά εμφανιζόμενη μεταξύ των 7 κοντινότρων γειτόνων. Ως μετρική για τον υπολογισμό των αποστάσεων μεταξύ των διανυσμάτων δοκιμάστηκε τόσο η ευκλείδεια απόσταση (Euklidean distance) όσο και η ομοιότητα συνημίτονων (Cosine similarity) με τη δεύτερη να πετυχαίνει ελαφρώς καλύτερα αποτελέσματα.

Grid Search στον ταξινομητή SVM:

Αναφορικά με τον SVM ταξινομητή, η επιλογή των υπερ-παραμέτρων γίνεται με τη χρήση της GridSearch συνάρτησης. Δεχόμενη ως όρισμα ένα Dictionary διαφορετικών επιλογών για τις C, Kernel και gamma παραμέτρους επιλέγεται ο συνδυασμός των τιμών των παραμέτρων για τον οποίο πετυχαίνουμε το μικρότερο σφάλμα. Η διαδικασία είναι χρονοβόρα οπότε έχει σχολιαστεί στη συνάρτηση main του αρχείου main.py και οι βέλτιστες τιμές που προέκυψαν από αυτή έχουν δοθεί ως υπερ-παράμετροι της SVM συνάρτησης που τελικά εκτελείται.

Αξιολόγηση αποτελεσμάτων με Kfold Cross Validation:

Για την αξιολόγηση της απόδοσης κάθε μεθόδου κατηγοριοποίησης υλοποιήθηκε μια K-fold-validation συνάρτηση. Η συνάρτηση αναλόγως της τιμής της παραμέτρου folds που στην υλοποίηση μας έχει τιμή 10, διαμερίζει τα δεδομένα προς κατηγοριοποίηση σε K set από τα οποία τα K-1 χρησιμοποιούνται για την εκπαίδευση του εκάστοτε κατηγοριοποιητή (training set) ενώ το 1 που απομένει αποτελεί το test set, και χρησιμοποιείται για τον έλεγχο της απόδοσης του κάθε αλγορίθμου. Η διαδικασία επαναλαμβάνεται K φορές και σε κάθε επανάληψη υπολογίζονται οι μετρικές Precision, Recall, F-measure και Accuracy βασιζόμενη στην i-οστή διαμέριση του data set ως test set και χρησιμοποιώντας τις υπόλοιπες για

εκπαίδευση. Με τον τρόπο αυτό όλες οι διαμερίσεις θα χρησιμοποιηθούν για test και ο τελικός υπολογισμός των μετρικών απόδοσης θα προκύψει ως ο μέσος όρος των επιμέρους υπολογισμών σε κάθε επανάληψη.

Πίνακας με τα αποτελέσματά μας:

Statistic Measure	**Naive Bayes**	**Random Forest**	**SVM**	**KNN**	**My Method**
Accuracy	95.92%	94.79%	96.72%	67.46%	97.31%
Precision	95.63%	94.70%	96.49%	67.35%	97.23%
Recall	95.52%	94.06%	96.53%	67.03%	97.24%
F-Measure	95.55	94.33%	96.50%	67.21%	97.22%

3.Beat the Benchmark

1)Συνδυασμός ταξινομητών:

Αναζητώντας το βέλτιστο ταξινομητή για το συγκεκριμένο πρόβλημα, χρησιμοποιήσαμε τις γνώσεις που αποκτήσαμε από το μάθημα “Αναγνώριση προτύπων – Μηχανική μάθηση”. Πιο συγκεκριμένα, σκεφτήκαμε να χρησιμοποιήσουμε συνδυασμό ταξινομητών για την ορθότερη πρόβλεψη των κατηγοριών. Η βασική φιλοσοφία πίσω από το συνδυασμό διαφορετικών ταξινομητών βασίζεται στο γεγονός ότι ακόμα και ο “καλύτερος” ταξινομητής αποτυγχάνει σε μερικά διανύσματα, όπου άλλοι ταξινομητές μπορεί να δώσουν σωστή ταξινόμηση. Ο συνδυασμός ταξινομητών σκοπεύει στην εκμετάλλευση αυτής της συμπληρωματικής πληροφορίας που δίνεται από διάφορους ταξινομητές.
Ο κανόνας με τον οποίο γίνεται η τελική πρόβλεψη είναι ο κανόνας της πλειοψηφίας, δηλαδή το διάνυσμα ταξινομείται στην κατηγορία που το κατατάσσουν οι περισσότεροι ταξινομητές, ενώ στην περίπτωση της ισοβαθμίας χρησιμοποιείται η πρόβλεψη του καλύτερου ταξινομητή (SVM).
Η αρχική μας προσέγγιση ήταν η χρήση τριών (3) ταξινομητών, δηλαδή συνδυασμός SVM, Linear SVM with SGD training (SGDClassifier) και Naive Bayes, η οποία παρουσίασε βελτίωση του accuracy κατά 0.3-0.5%. Πειραματιζόμενοι με συνδυασμό άλλων ταξινομητών και στοχεύοντας στην όσο το δυνατό μεγαλύτερη ανεξαρτησία των ταξινομητών που επιλέγουμε, καταλήξαμε πως το καλύτερο accuracy (συνολική αύξηση 0.6-1%) επιτυγχάνεται με τον συνδυασμό SVM, Naive Bayes και Νευρωνικό δίκτυο (MLPClassifier) με 8 κρυφά επίπεδα και 10 νευρώνες σε κάθε επίπεδο.

2)Stemming:

Αναφορικά με την προεπεξεργασία των δεδομένων, προσπαθήσαμε να εφαρμόζουμε stemming, δηλαδή να απλουστεύσουμε όσο γίνεται τα διανύσματα, συμπτύσσοντας τις λέξεις με κοινή ρίζα σε μία. Για παράδειγμα, οι λέξεις “fishing”, “fished” και “fisher” αντικαθίστανται από τη λέξη “fish”.

Η διαδικασία του stemming πραγματοποιείται κατά το Vectorization των προτάσεων. Προσπαθήσαμε να χρησιμοποιήσουμε stemmer από τις βιβλιοθήκες SpaCy και NLTK (PorterStemmer και SnowballStemmer), αλλά και στις δύο περιπτώσεις το accuracy ήταν χαμηλότερο από αυτό χωρίς stemming. Ενδεικτική χρήση του stemming φαίνεται στο αρχείο “classifiers_and_stemming.py”.