Τεχνικές Εξόρυξης Δεδομένων

Εργασία 1

- Νεαμονίτης Ευάγγελος, **Α.Μ.:** 1115201400123
- Ορφανίδης Ελευθέριος, **Α.Μ.:** 1115201400133

Αρχεία:

wordclouds.py: Το αρχείο με τον κώδικα για την παραγωγή των Wordclouds.

knn_algorithm.py: Περιέχει την υλοποίηση μας του knn και το cross validation.

svm.py: Περιέχει το GridSearch και το CrossValidation για τον svm.

random_forests.py: Περιέχει το GridSearch και το CrossValidation για τον random forests.

nb.py: Περιέχει το GridSearch και το CrossValidation για τον Multinomial NB.

my_method.py: Περιέχει το prediction για το test set και την παραγωγή του αρχείου testSet_categories.csv.

my_method_10_fold.py: Περιέχει μόνο το cross validation για την καλύτερη μέθοδο.

ten_fold_cross.py: Το αρχείο αυτό καλεί όλα τα 10-fold cross validation των αλγόριθμων και εξάγει το EvaluationMetric_10fold.csv

Σημ: οι παράμετροι που είναι στον κώδικα των GridSearch ΔΕΝ είναι οι μόνες που δοκιμάσαμε. Είναι ενδεικτικές.

Ερώτημα 1 - Wordcloud

Για την δημιουργία των Wordcloud ακολουθούμε την εξής διαδικασία:

1. Διαβάζουμε τα δεδομένα του train_set και τα περνάμε σε μια μεταβλητή train_data
2. Περνάμε σε ξεχωριστές μεταβλητές τα άρθρα ανάλογα με την κατηγορία στην οποία βρίσκονται (πχ. Στην μεταβλητή dataPol όλα τα άρθρα του train_data τα οποία ανήκουν στην κατηγορία Politics κλπ.)

3. Στην συνέχεια χρησιμοποιούμε αυτές τις μεταβλητές (αφού τις μετατρέψουμε σε string) ως είσοδο στην συνάρτηση Wordcloud χρησιμοποιώντας αρχικά για stopwords το σύνολο ENGLISH_STOP_WORDS που προσφέρει το scikit learn.
4. Τέλος ελέγχουμε τα wordclouds που δημιουργήθηκαν και προσθέτουμε στα stopwords λέξεις που εμφανίζονται ως πιο συχνές σε παραπάνω από ένα wordcloud.

Ερώτημα 2 - Classification

Προ-επεξεργασία

Στο ερώτημα 2 για την προ-επεξεργασία των δεδομένων έγιναν τα εξής βήματα:

- Σε επίπεδο text η προ-επεξεργασία που κάνουμε είναι η αφαίρεση των βασικών stopwords της αγγλικής γλώσσας μαζί με κάποιες λέξεις που παρατηρήσαμε από τα wordclouds ότι εμφανίζονται πολύ συχνά σε όλες τις κατηγορίες.

- Στην συνέχεια με την χρήση του CountVectorizer μετατρέπεται το κείμενο σε διάνυσμα τα στοιχεία του οποίου είναι ο αριθμός των φορών που βρέθηκε η κάθε λέξη του κειμένου προκειμένου να δοθεί σαν όρισμα στον classifier.

- Μετά κανονικοποιείται με την χρήση της συνάρτησης TfidfTransformer όπου δίνει βαρύτητα σε κάθε λέξη ανάλογα με τη συχνότητα εμφάνισής της.

- Τέλος για του αλγορίθμους SVM , RandomForests, KNearestNeighbor κάνουμε και περικοπή της διάστασης των διανυσμάτων που δημιουργήθηκαν, με την συνάρτηση TruncatedSVD.

Στη συνέχεια γίνεται το cross-validation:

- Αρχικά τα folds δημηουργούνται με την χρήση της StratifiedKFold καθώς το μοίρασμα που κάνει για κάθε fold περιέχει ίσο αριθμό από αρχεία της ίδιας κλάσης.
- Στη συνέχεια φτιάχνουμε ένα pipeline που περιέχει τα παραπάνω transforms και τον classifier που χρησιμοποιείται σε κάθε περίπτωση, με σκοπό την ομαδοποίηση των εντολών αυτών.
- Για το Cross Validation δεν χρησιμοποιήσαμε κάποια συνάρτηση όπως η cross_validate ή την cross_val_score και χρησιμοποιήσαμε iterators ώστε να έχουμε πλήρη έλεγχο σε κάθε fold και να μπορούμε να βλέπουμε και να διαχειριζόμαστε τα scores από το καθένα ξεχωριστά .

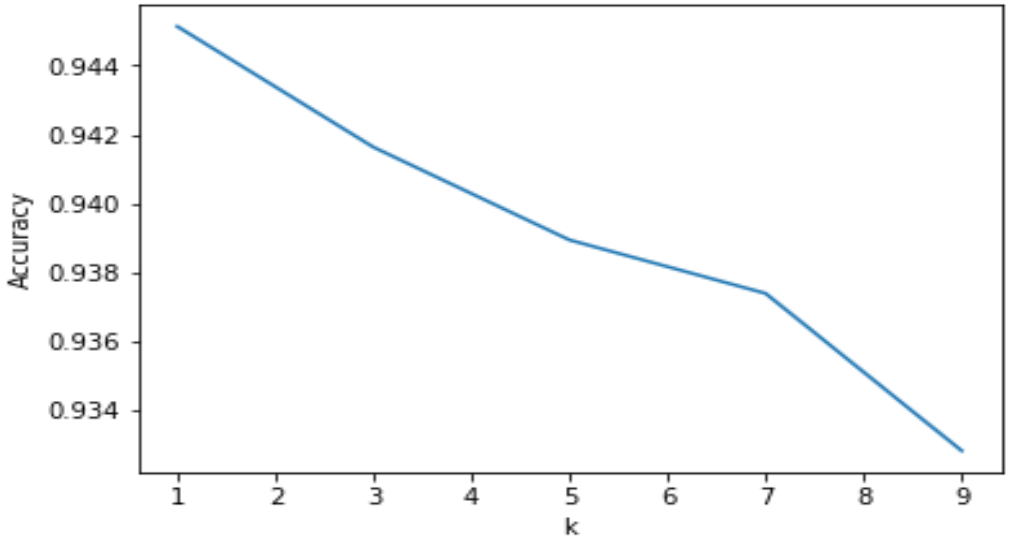
Αλγόριθμος K nearest neighbors

Για την υλοποίηση του αλγορίθμου των Κ κοντινότερων γειτόνων δημιουργήσαμε μία κλάση η οποία, όταν καλείται, δέχεται (προαιρετικά) ένα όρισμα το οποίο ορίζει τον k (τον αριθμό των κοντινότερων γειτόνων από τους οποίους θα γίνει το majority voting). Η υλοποίηση μας μπορεί να γίνει import και να χρησιμοποιηθεί κανονικά όπως οποιαδήποτε άλλη συνάρτηση κατηγοριοποίησης .
Συγκεκριμένα όταν καλείται η knn.fit(X, y) αποθηκεύουμε στις μεταβλητές της κλάσης self.X_train και self.y_train το X και το y αντίστοιχα.
Η knn.predict(X_test) επιστρέφει μια λίστα με τις προβλέψεις που έκανε ο αλγόριθμος (με το scores[0] να αντιστοιχεί στην πρόβλεψη του πρώτου στοιχείου του X_test, του scores[1] του 2ου κ.ο.κ..
Ο τρόπος λειτουργίας της knn.predict(X_test) ειναι:

1. Διατρέχει με ένα for όλα τα στοιχεία του X_test
2. Για κάθε στοιχείο καλεί την συνάρτηση find_Knn(item) η οποία:
    a. Διατρέχει όλο το X_train και για κάθε στοιχείο του, μετράει την απόσταση που έχει απο το item
    b. Κάνει append την απόσταση στην λίστα distances
    c. Αφού διατρέξει όλο το X_train βρίσκει στην λίστα distances τις k μικρότερες τιμές και εν συνεχεία τα k διανύσματα που απέχουν λιγότερο απο το item.
    d. Τέλος επιστρέφει την κλάση που έχει περισσότερους “εκπροσώπους” σε αυτά τα k διανύσματα. (κάνει δηλαδή majority voting).
3. Κάνει append τον αριθμό της κλάσης που επεστράφη στην λίστα predictions
4. Μόλις ολοκληρωθεί η διαδικασία, επιστρέφει την λίστα predictions.

Επιλογή k (GridSearch)

Πραγματοποιήσαμε 10-fold cross-validation για διαφορετικές τιμές του k ώστε να βρούμε ποιο είναι πιο αποτελεσματικό και λάβαμε τα εξής αποτελέσματα:



Γράφημα accuracy - k_neighbors

Όπως παρατηρούμε το μεγαλύτερο accuracy το πετυχαίνουμε με τιμή k = 1 (δηλαδή χρειάζεται απλά να βρούμε τον κοντινότερο γείτονα κάποιου στοιχείου για να το κατηγοριοποιήσουμε. Άρα τυπικά δεν ακολουθείται η λογική του majority voting με k = 1 καθώς δεν υπάρχουν αρκετοί γείτονες ώστε να επιλέξουμε την κλάση της πλειοψηφίας).

Αποτελέσματα 10-fold cross-validation:

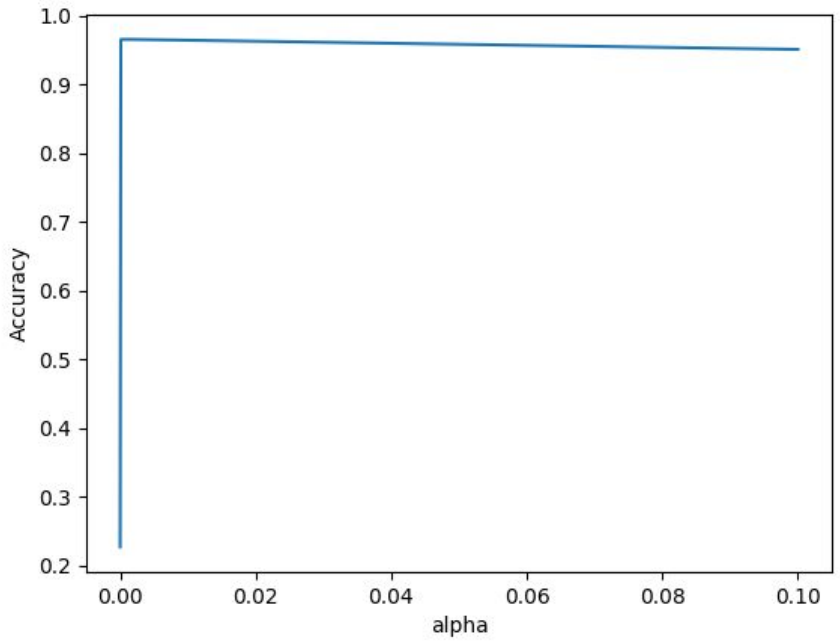
Average accuracy: 0.9368
Average recall : 0.9363
Average f_score : 0.9342
Average precision : 0.9337

Multinomial Naive-Bayes

Στην προεπεξεργασία των δεδομένων χρησιμοποιήσαμε μόνο τον Countvectorizer με μόνα ορίσματα τα stopwords και το max_features = 1000 (ώστε να έχει μικρότερες διαστάσεις το κάθε διάνυσμα που θα γίνει vectorize). Στην συνέχεια χρησιμοποιούμε TfidfTransformer και τα δεδομένα μας είναι έτοιμα για το cv. Τα αποτελέσματα του 10-fold cross-validation με την χρήση του ταξινομητή Multinomial Naive Bayes είναι τα εξής:

Average Accuracy: 0.91162
Average Precision: 0.92500
Average Recall: 0.87399
Average Fscore: 0.88267

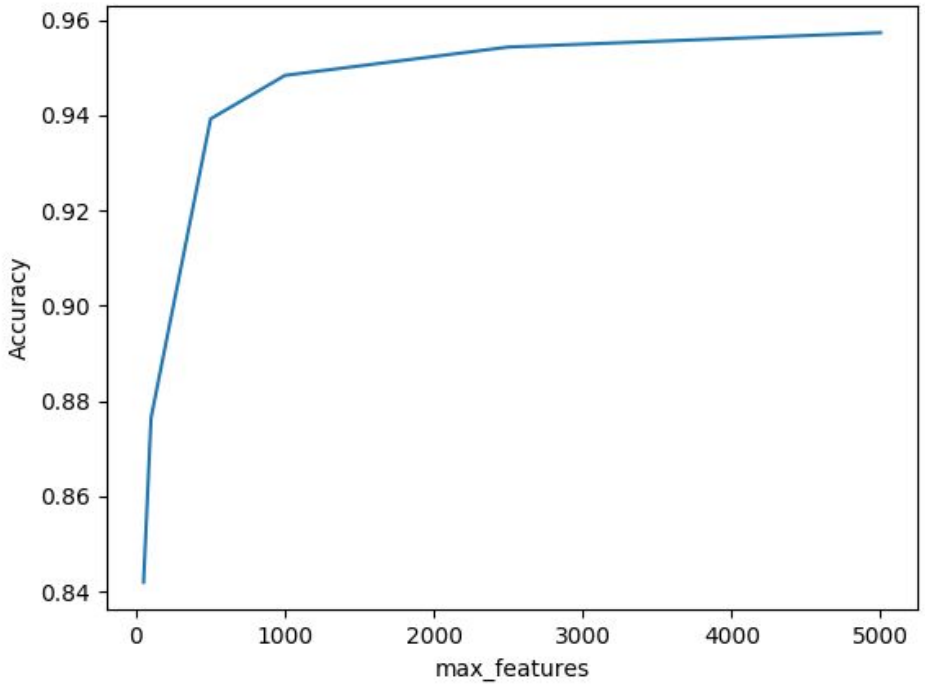
GridSearch για alpha:



Όπως βλέπουμε και στο plot η πιο αποδοτική εκτέλεση του αλγόριθμου έγινε με alpha = 0.0001 πετυχαίνοντας accuracy = 0.966 και τα αποτελέσματα που πήραμε ήταν τα παρακάτω:

Average Accuracy: 0.95556
Average Precision: 0.95197
Average Recall: 0.95208
Average Fscore: 0.95178

GridSearch για max features:



Όπως παρατηρούμε στο plot, όσο αυξάνουμε το max_features τόσο αυξάνεται και η ακρίβεια του αλγορίθμου. Αλλά η αύξηση της ακρίβειας από τα 2500 features και πάνω είναι μικρή, οπότε για το ερώτημα 2 επιλέξαμε ως max_features τα 2500 καθώς από 5000 και πάνω έχουμε αισθητή καθυστέρηση του αλγορίθμου.

Support Vector Machines (SVM)

Για το ερώτημα αυτό όπως και πριν στην προ-επεξεργασία των δεδομένων χρησιμοποιήσαμε μόνο τον Countvectorizer με μόνο ορίσματα τα stopwords . Στην συνέχεια χρησιμοποιούμε TfidfTransformer και SVD για τη μείωση της διάστασης με τον προτεινόμενο από το documentation αριθμό components (100). Εδώ ακόμα δεν έχει γίνει GridSearch για τις παραμέτρους του αλγορίθμου. Τα αποτελέσματα του 10-fold cross-validation με την χρήση του ταξινομητή SVM είναι τα εξής:

Average Accuracy: 0.87311
Average Precision: 0.89208
Average Recall: 0.827971
Average F1 score: 0.82698

GridSearch για C, gamma, kernel:

Τρέχοντας διάφορες δοκιμές και μεγάλο εύρος τιμών για τα C και gamma τα καλυτερα αποτελέσματα που πήραμε ήταν:

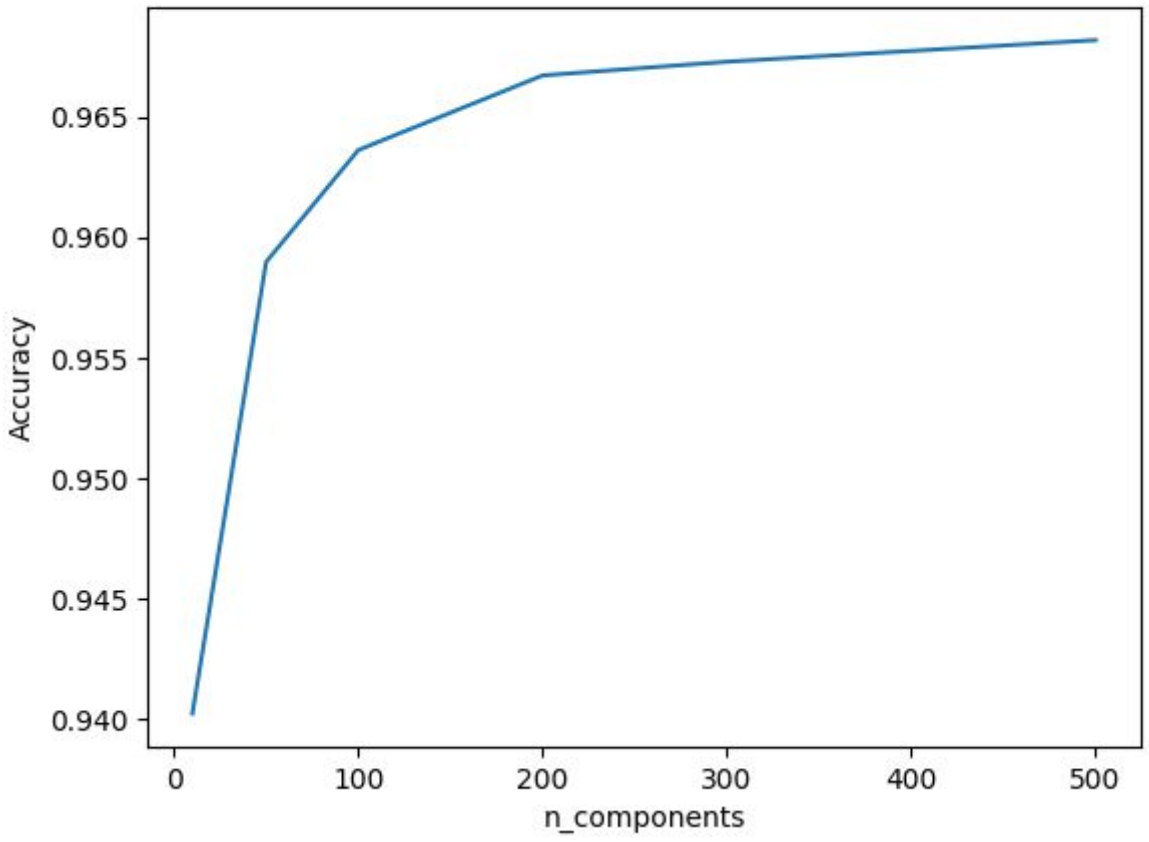
C=10000
gamma = 0.0001
kernel = ‘rbf’

και τα αποτελέσματα του 10-fold cross validation ήταν τα παρακάτω

Average Accuracy: 0.96306
Average Precision: 0.96034
Average Recall: 0.96014
Average F1 score: 0.96013

GridSearch για n components:

Κάνοντας GridSearch με τον αριθμό των components για το SVD πήραμε το παρακάτω γράφημα:



Βλέπουμε πως από το 300 και μετά δεν υπάρχει σημαντική αύξηση του accuracy για αυτό και κρατήσαμε την τιμή αυτή. Παρόλο που αργούσε λίγο παραπάνω έδινε αρκετά καλύτερα αποτελέσματα από 100 components και για παραπάνω από 300 αργούσε σημαντικά χωρίς να δίνει καλύτερα αποτελέσματα.

Average Accuracy: 0.96592
Average Precision: 0.96373
Average Recall: 0.96379
Average F1 score: 0.96362

Random Forests

Και για τον αλγόριθμο αυτό ξεκινήσαμε χρησιμοποιώντας μόνο τον CountVectorizer με μόνο ορίσμα τα stopwords . Στην συνέχεια χρησιμοποιούμε TfidfTransformer και SVD για τη μείωση της διάστασης με τον προτεινόμενο από το documentation αριθμό components (100). Εδώ ακόμα δεν έχει γίνει GridSearch για τις παραμέτρους του αλγορίθμου. Τα αποτελέσματα του 10-fold cross-validation με την χρήση του ταξινομητή Random Forests είναι τα εξής:

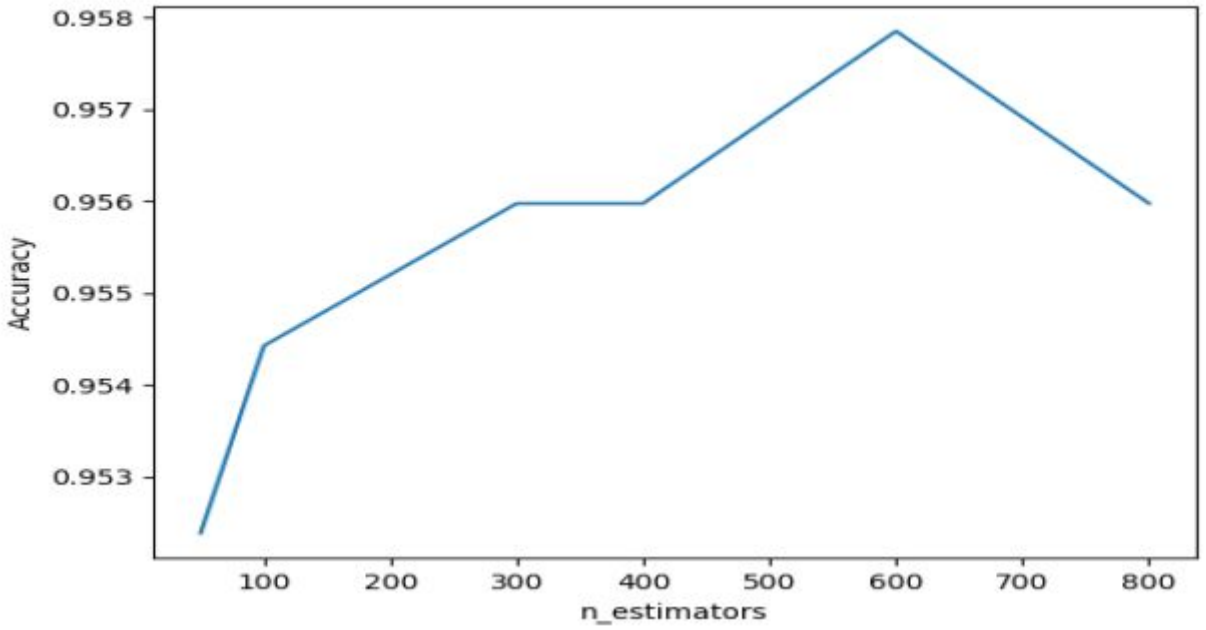
Average Accuracy: 0.93803
Average Precision: 0.94662
Average Recall: 0.93161
Average F1 score: 0.93278

GridSearch για max_depth, min_samples_leaf, n_estimators:

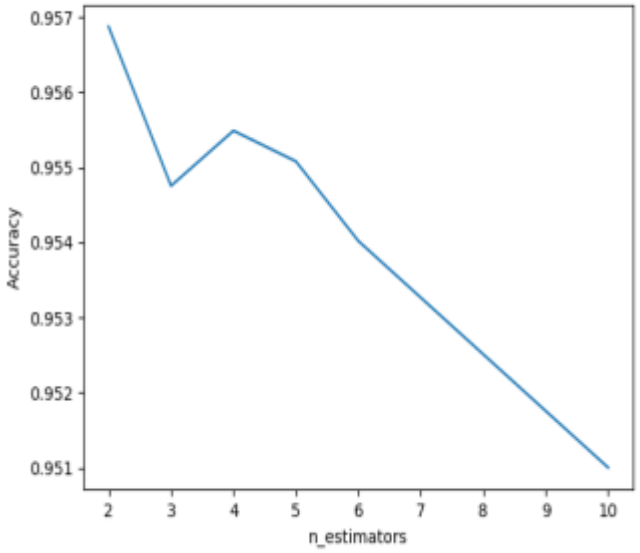
Όπου:

- n_estimators: ο αριθμός από δέντρα που δημιουργούνται
- max_depth: το βάθος του κάθε δέντρου
- min_samples_leaf: ο ελάχιστος αριθμός δειγμάτων που απαιτείται να βρίσκεται σε ένα κόμβο-φύλλο

Γράφημα 1: Σχέση accuracy - n_estimators

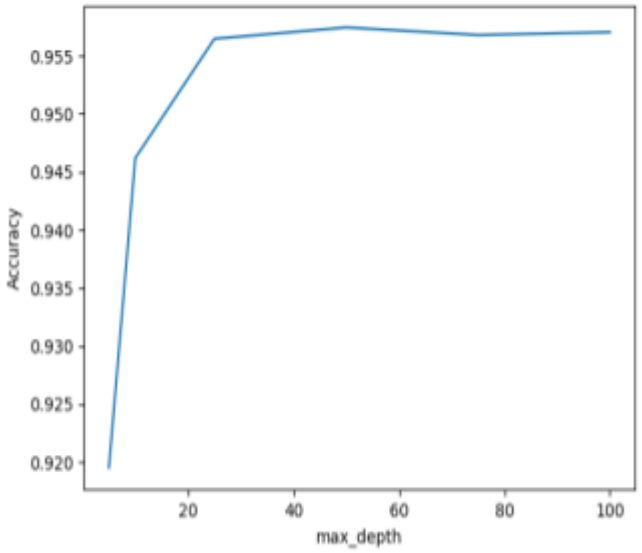


Γράφημα 2: Σχέση accuracy - min_samples_leaf



*με n_estimators = 600

Γράφημα 3: Σχέση accuracy - max_depth



*με n_estimators = 600 και min_samples_leaf = 2

Οι καλύτερες τιμές που πήραμε ήταν οι εξής:

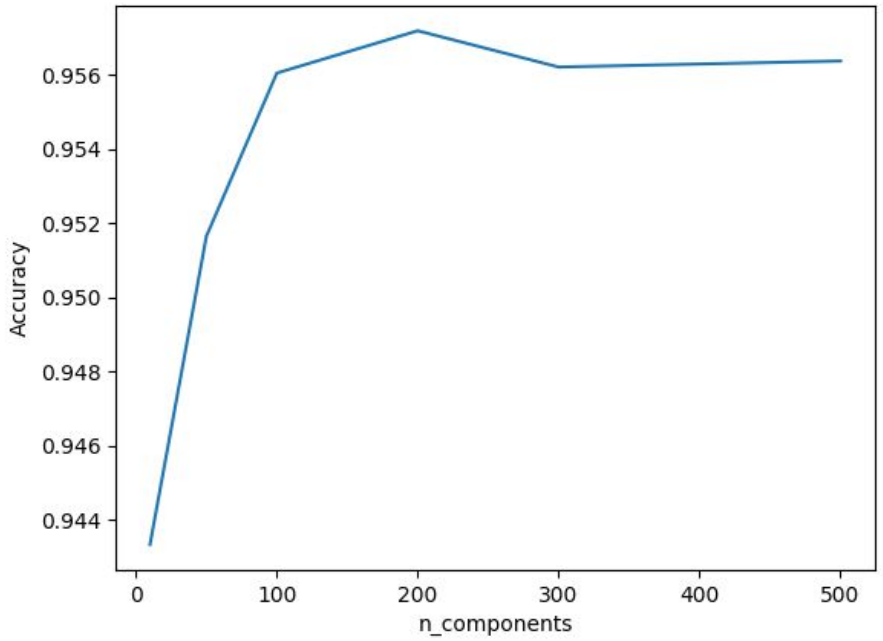
- n_estimators = 600
- min_samples_leaf = 2
- max_depth = 50

Τα αποτελέσματα του RandomForests Classifier (με τις optimal τιμές στις παραμέτρους του και 100 components) είναι:

Average Accuracy: 0.95442
Average Precision: 0.95107
Average Recall: 0.95012
Average F1 score: 0.95042

GridSearch για n components:

Κάνοντας GridSearch με τον αριθμό των components για τον Random Forests πήραμε το παρακάτω γράφημα:



και έτσι χρησιμοποιώντας 200 components τα τελικά αποτελέσματα που πήραμε ήταν:

Average Accuracy: 0.95605
Average Precision: 0.95330
Average Recall: 0.95157
Average F1 score: 0.95219

Ερώτημα 3 - Beat the benchmark

Τα πρώτα απλά βήματα τα κάναμε από το ερώτημα 2:

- Πρόσθεση στη λίστα των stopwords (said, says, saying)
- GridSearch για καλύτερες παραμέτρους

Ξεκινώντας κάναμε μια απλή αξιοποίηση του τίτλου βάζοντας τον στην αρχή του Content ώστε να υπολογιστεί και αυτός στην δημηουργία του vecor:

```
train_data["Content"]=(train_data["Title"])+" "+(train_data["Content"])
test_data["Content"]=(test_data["Title"])+" "+(test_data["Content"])
```

Δοκιμάσαμε να πάρουμε καλύτερη απόδοση από τους αλγορίθμους που δοκιμάσαμε στο πρώτο ερώτημα. Ο καλύτερος αλγόριθμος από το ερώτημα 2 ήταν ο SVM και συνεπώς ξεκινήσαμε με αυτόν.

Η πρώτη ιδέα που είχαμε ήταν να παρατηρήσουμε τα wordclouds και να αφαιρέσουμε λέξεις που ήταν κοινές σε αρκετές κατηγορίες κειμένων. Καταλήγοντας στην πρόσθεση της λίστας ["said","say","says","just","did","was","were","year","years","like","people"] βλέποντας μια πολύ μικρή βελτίωση.

Στη συνέχεια παρατηρήσαμε πως οι λέξεις που έχουν σημεία στίξης μετρώνται σαν διαφορετικές και επομένως αφαιρέσαμε όλα τα σημεία στίξης χρησιμοποιώντας τη συνάρτηση strip_punctuation πριν αφαιρέσουμε τα stopwords. το accuracy βελτιώθηκε επίσης ελάχιστα. ΣΗΜ: κάναμε όλες τις επεξεργασίες σε unicode καθώς αντιμετωπίζαμε προβλήματα με χαρακτήρες σε ascii.

Δοκιμάσαμε επίσης να μετατρέψουμε όλες τις χρηματικές ποσότητες σε μια λέξη moneyamount ώστε να υπολογίζονται σαν μια λέξη ανεξάρτητα από το ποσό, με το σκεπτικό πως κάτι τέτοιο θα βοηθούσε στην αναγνώριση κατηγοριών που περιέχουν αρκετά ποσά. Κάτι τέτοιο όμως δεν έκανε καμία διαφορά και συνεπώς είναι σε σχόλια

Σε επόμενο βήμα δοκιμάσαμε να πειραματιστούμε με stemming και lemmatization.

1. Αρχικά διαβάζοντας το documentation που δόθηκε στην άσκηση (https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html) είδαμε ότι το lemmatization κάνει αρκετά μεγαλύτερη επεξεργασία από ότι το stemming σε ότι αφορά τη σημασία των λέξεων και έτσι ξεκινήσαμε να το κάνουμε στα κείμενα. Όμως ο χρόνος που έκανε για να τρέξει σε όλα τα δεδομένα ήταν πάρα πολύ μεγάλος (σε συνδυασμό κιόλας με την αφαίρεση σημείων στίξης και stopwords) χωρις να προσφέρει μεγάλες διαφορές στο accuracy. Ο κώδικας για το lemmatization επομένως είναι σε σχόλια στο αρχείο my_ method.py

2. Στη συνέχεια δοκιμάσαμε τον PorterStemer από την βιβλιοθήκη gensim. παρά το γεγονός οτι η επεξεργασία που κάνει είναι πιο “πρόχειρη” σε σχέση με το lemmatization ο χρόνος εκτέλεσης του είναι σημαντικά μικρότερος και τα αποτελέσματα δεν είχαν μεγάλη απόκλιση και έτσι αποφασίσαμε να κρατήσουμε αυτό στην τελική μας υλοποίηση.

Με την καλύτερη επεξεργασία και μετά από αρκετό ψάξιμο με τον SVM τα καλύτερα αποτελέσματα που καταφέρουμε να πετύχουμε στο kaggle ήταν 0.96739 και έτσι αρχίσαμε να δοκιμάζουμε και τους υπόλοιπους αλγόριθμους.

- Ο Random Forests δεν έδωσε κάποια ιδιαίτερη απόδοση παρά τα διαφορετικά GridSearch που έγιναν στις πολλές παραμέτρους του με μεγάλο εύρος τιμών. Έτσι αποφασίσαμε να μην ασχοληθούμε περισσότερο με αυτόν.

- Για τον Multinomial Naive Bayes αποφασίσαμε να βάλουμε το όρισμα ngram_range = (1,2) στον CountVectorizer ώστε να διευρύνουμε το λεξιλόγιο που θα είχε στην διάθεση του ο αλγόριθμος. Με αυτή την αλλαγή καταλήξαμε σε αποτελέσματα που αποτελούν και την καλύτερη επίδοση που καταφέραμε να πετύχουμε με την χρήση του MultinomialNB() στο kaggle 0.96847 και την επεξεργασία κειμένου που περιγράφεται παραπάνω. Τα αποτελέσματα του όμως στο 10-fold cross validation δεν ήταν καλύτερα συνολικά:

  Total Accuracy: 0.962252820885
  Total Precision: 0.959403989897
  Total Recall 0.958692789534
  Total F1 score 0.958949145386

Σε μια προσπάθεια να κάνουμε κάτι ακόμα καλύτερο και αφού είδαμε ότι μπορούμε να χρησιμοποιήσουμε και άλλους αλγόριθμους πέρα από αυτούς που καλούμαστε να υλοποιήσουμε για το ερώτημα 2 αρχίσαμε να ψάχνουμε για Multiclass classification και από το documentation του sklearn αρχίσαμε να δοκιμάζουμε αλγόριθμους από αυτή τη σελίδα: http://scikit-learn.org/stable/modules/multiclass.html. Επειδή όμως δεν προλαβαίναμε να τρέξουμε GridSearch και να πειραματιστούμε αρκετά με όλους τους αλγόριθμους δοκιμάσαμε τον LinearSVC και τον Logistic Regression CV που έδωσαν καλά αποτελέσματα αρκετά σύντομα.

Παρόλα αυτά συνεχίζαμε να παίρνουμε τα καλύτερα αποτελέσματα από τον SVM και επομένως πειραματιστήκαμε με τις συναρτήσεις OneVsOne η οποία κάνει fit έναν classifier ανά ζευγάρι κλάσεων και OneVsRest που κάνει fit έναν classifier ανά κλάση και τον SVM με τις παραμέτρους που είχαμε από το GridSearch.

Τελικά τα καλύτερα αποτελέσματα στο 10-fold cross validation τα πήραμε χρησιμοποιώντας την OneVsRest με τον SVM και αποφασίσαμε να χρησιμοποιήσουμε αυτό για το ερώτημα 3.

Η τελευταίες αλλαγές που έγιναν και μας έδωσαν και τα καλύτερα συνολικά αποτελέσματα ήταν να μην χρησιμοποιήσουμε LSI καθώς και να χρησιμοποιήσουμε το ngram_range = (1,2) ως όρισμα στον vectorizer ώστε να διευρυνθεί το λεξιλόγιο που έχει στην διάθεση του ο αλγόριθμος . Αυτό μας έδωσε στα leaderboards την τιμή 0.97024 και συνολικά τα καλύτερα αποτελέσματα στο 10-fold cross validation:

Average Accuracy: 0.96909
Average Precision: 0.96701
Average Recall: 0.96702
Average F1 score: 0.96694