DATA MINING TECHNIQUES

FIRST ASSIGNMENT

Lida Zacharopoulou AM 1115201100004
Seintaridis Dimitrios AM 1115201100197

Προεπεξεργασία δεδομένων - WordClouds

Για την προεπεξεργασία δεδομένων αφαιρέσαμε αρχικά από το συνολικό λεξιλόγιο(corpus) τα stopwords(λέξεις που δεν προσφέρουν κάποια σημαντική πληροφορία στον ταξινομητή καθώς είναι πολύ συχνές). Τέτοιες λέξεις για παράδειγμα ειναι το is,I,the,κτλ. Για να το κάνουμε αυτό χρησιμοποιήσαμε τα stopwords ENGLISH_STOP_WORDS της βιβλιοθήκης sklearn.feature_extraction.text

Επιπλέον, μετά από παρατηρήσεις πάνω στα wordcloud images που παράγουμε, επιλέξαμε να προσθέσουμε κάποια additional stopwords με λέξεις που δεν πρόσφεραν κάποια αξία στο αποτέλεσμα, αλλά εμφανίζονταν με μεγάλη συχνότητα όπως said, much κτλ

Προεπεξεργασία δεδομένων - Παράδειγμα Football

WordCloud με ENGLISH_STOP_WORDS

WordCloud με ENGLISH_STOP_WORDS + additional stopwords

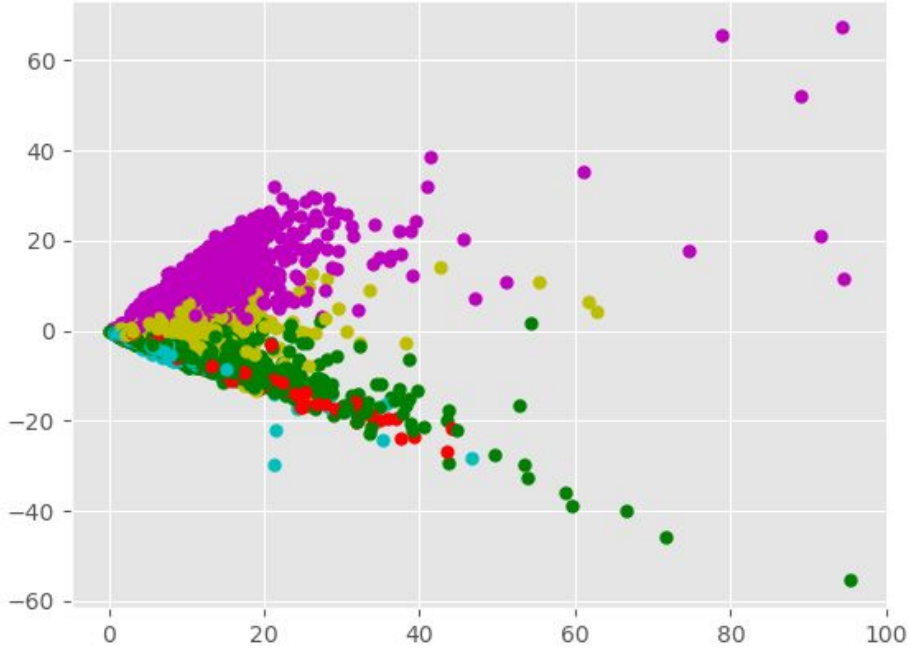
Προεπεξεργασία δεδομένων - Vectorizers

Για να μπορέσουμε να επεξεργαστούμε τα δεδομένα τα μετατρέψαμε σε πίνακες αριθμών με την χρήση vectorizer, συγκεκριμένα TfidfVectorizer και CountVectorizer

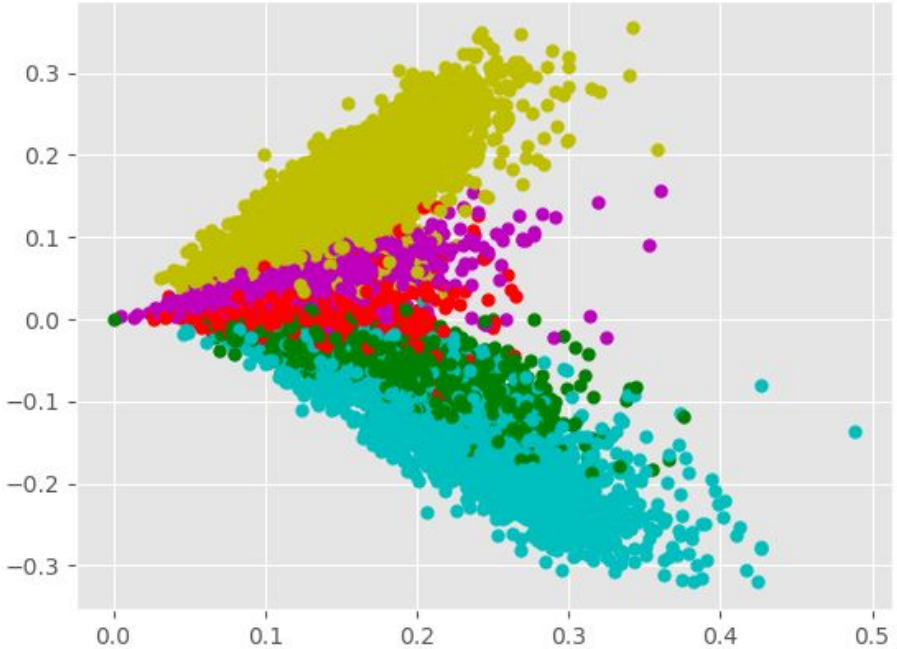
Από τις μετρήσεις μας επιλέξαμε των TFIDF vectorizer

Clustering - Counter vs TFIDF Vectorizer

nltk kmeans, Cosine Similarity,
Counter Vectorizer

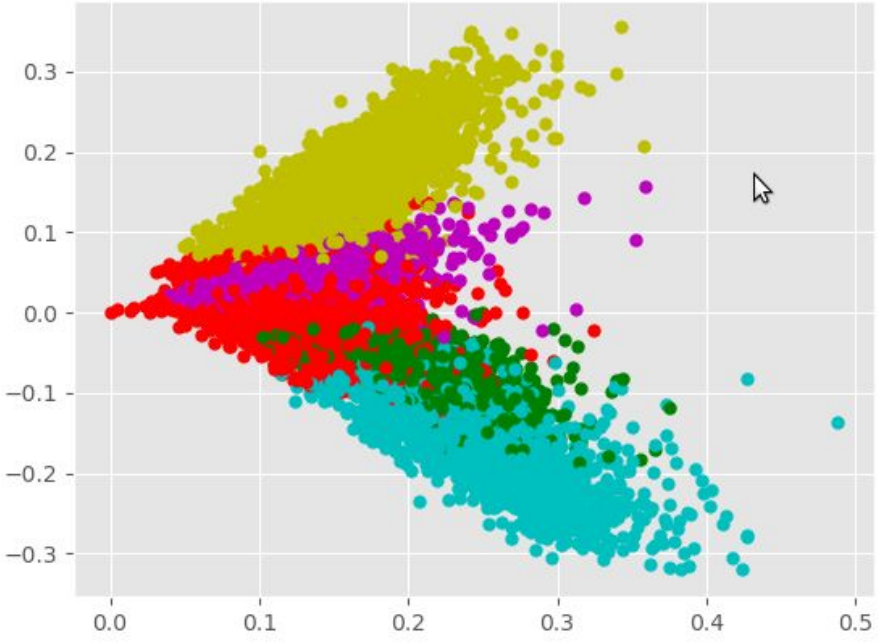


nltk kmeans, Cosine Similarity,
TFIDF Vectorizer

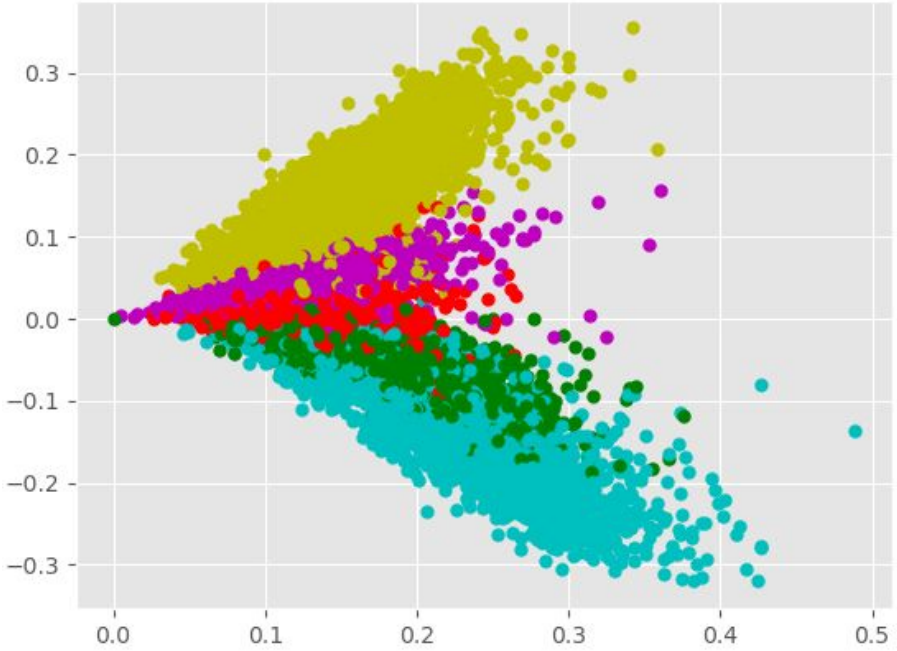


Clustering - Euclidean Distance vs Cosine Similarity

nltk kmeans, TFIDF Vectorizer,
Euclidean Distance



nltk kmeans, TFIDF Vectorizer,
Cosine Similarity



Clustering

Σύγκριση αποτελεσμάτων

nltk kmeans, Cosine Similarity, Preproccessing με TfidfVectorizer και LSI

	Politics	Football	Technology	Film	Business
Cluster1	0.12	**0.86**	0	0.01	0.01
Cluster2	0.03	0.31	0.01	**0.64**	0.01
Cluster3	**0.97**	0.02	0	0	0.01
Cluster4	0	0	0	0	**0.99**
Cluster5	0.03	0.01	**0.89**	0.05	0.01

Μετά από μετρήσεις και δοκιμές, καταλήξαμε στο ότι η υλοποίηση με τα καλύτερα αποτελέσματα είναι αυτή, όπως φαίνεται και στον πίνακα παραπάνω (αποτελέσματα απο clustering_KMeans.csv

Clustering

nltk kmeans, Cosine Similarity, Preproccessing με **CountVectorizer** και LSI

	Politics	Football	Technology	Film	Business
Cluster1	0.05	0.02	**0.71**	**0.21**	0.01
Cluster2	0.01	0	0.03	0.13	**0.82**
Cluster3	0	0	0	0	**0.99**
Cluster4	0.15	**0.65**	0.01	0.18	0.01
Cluster5	**0.97**	0.02	0	0	0.01

Παρατηρούμε ότι η υλοποίηση αυτή δεν είναι καθόλου αποτελεσματική καθώς στο και στο Cluster2 και στο Cluster3 συγκεντρώνονται λέξεις της κατηγορίας Business, ενώ η κατηγορία Film εμφανίζει το μεγαλύτερο ποσοστό της (μόλις 21%) στο Cluster1, το οποίο όμως κατα 71% αποτελείται από λέξεις της κατηγορίας Technology.

Clustering

nltk kmeans, **Euclidean Distance**, Preproccessing με TfidfVectorizer και LSI

	Politics	Football	Technology	Film	Business
Cluster1	0.1	**0.9**	0	0	0
Cluster2	0.22	0.32	0.02	**0.35**	0.09
Cluster3	**0.99**	0.01	0	0	0
Cluster4	0	0	0	0	**1**
Cluster5	0.02	0	**0.94**	0.03	0.01

Η υλοποίηση αυτή δεν είναι η βέλτιστη καθώς η κατηγορία Film εμφανίζεται σε μεγαλύτερα ποσοστά στο Cluster2, αλλά το ποσοστό αυτό είναι πάρα πολύ μικρό (35%) και πολύ κοντά με το ποσοστό των λέξεων της κατηγορίας Football στο ίδιο cluster

Clustering

- Για το clustering χρησμοποιήσαμε την υλοποίηση του kmeans που υπάρχει στην βιβλιοθήκη nltk της Python, ώστε να μπορούμε να δώσουμε τη συνάρτηση απόστασης ως όρισμα.

- Για την cosine similarity ως συνάρτηση απόστασης χρησιμοποιήθηκε η cosine distance από το nltk.cluster (1 - cosine distance = cosine similarity)

- Αυξάνοντας τον αριθμό των επαναλήψεων του kmeans, παρατηρήσαμε ότι δεν πετυχημένουμε καλύτερα αποτελέσματα, ενώ ο χρόνος εκτέλεσης αυξάνεται σημαντικά. Ενδεικτικά, οι χρόνοι εκτέλεσεις ήταν:
  Για n = 500 t ~ 20min,
  Για n = 100 t ~ 5min
  Για n = 25 t ~ 1min

  Επομένως επιλέξαμε το n=25 για αριθμό επαναλήψεων

- Παρατηρήσαμε τέλος ότι η η υλοποίηση του kmeans της βιβλιοθήκης sklearn είχε τα ίδια αποτελέσματα με αυτή της βιβλιοθήκης nltk χρησιμοποιώντας την Euclidean distance ως συνάρτηση απόστασης, αλλά ο χρόνος εκτέλεσης ήταν αρκετά μικρότερος.

Classification

Ζητούμενο:

1. Εύρεση καλύτερου ταξινομητή ύστερα από μετήσεις
2. Κατηγοροποίηση κειμένων με την χρήση διαφορετικών μεθόδων ταξινόμησης και μετρικών.
3. Βελτίωση ταξινομητή

Περιγραφή εργασίας(1)

Για το classification δοκιμάσαμε τις παρακάτω μεθόδους:

- Support Vector Machines(SVM)
- Random Forests
- Naive Bayes
- K-Nearest Neighbor(δική μας υλοποίηση)

Για να αξιολογήσουμε και να καταγράψουμε την απόδοση κάθε μεθόδου χρησιμοποιήσαμε 10-fold Cross Validation χρησιμοποιώντας τις παρακάτω μετρικές:

- Precision / Recall / F-Measure
- Accuracy
- AUC
- ROC plot

Περιγραφή εργασίας(2)

Κατά την προ-επεξεργασία των δεδομένων χρησιμοποιήσαμε την τεχνική “Latent Semantic Indexing (LSI)” δοκιμάσαμε διαφορετικό αριθμό components κρατώντας σταθερό τον καλύτερο μας ταξινομητή(SVM). Το γράφημα παρουσιάζεται και παρακάτω στις διαφάνειες αλλά και στο φακελο output με το όνομα LSIcomponentsAccuracy.png το οποίο παράγεται εκτελώντας μια συνάρτηση στο πρόγραμμα μας.

Προσπαθήσαμε να χρησιμοποιήσουμε αποδοτικά τον τίτλο δίνοντας του διαφορετικά βάρη χωρίς τρομερή βελτίωση. Τα διαφορετικά βάρη μπορούμε να τα περάσουμε σαν παράμετρο στις συναρτήσεις μας.

Περιγραφή εργασίας(3)

Για K-Nearest Neighbor δεν χρησιμοποίησαμε έτοιμη υλοποίηση αλλά τον υλοποιήσαμε μόνοι μας. Χρησιμοποίσαμε έτοιμη την ευκλίδεια απόσταση ωστόσο για να μετράμε τις αποστάσεις των σημείων.Η επιλογή του τελικού label γίνεται με Majority voting.

Για τον SVM πειραματιστήκαμε με τις παραμέτρους kernel(rbf η default του sklearn,linear), c και gamma.

Αρχεία Εξόδου στον φάκελο output

EvaluationMetric_10fold.csv περιέχει τους μέσους όρους των μετρήσεων για όλες τις μεθόδους ταξινόμησης και όλες τις μετρικές

testSet_categories.csv περιέχει τις προβλέψεις μου για το test dataset

Ολα τα αρχεία category_roc περιέχουν τα διαγράμματα roc για κάθε κατηγορία για κάθε fold.

Cross validation results

Για το cross validation χρησιμοποίησα nfold=10. Τα αποτελέσματα παρακάτω είναι το μεσόρο των τιμων όλων τών folds. Καλύτερος ταξινομητής βγήκε ο SVM με kernel=linear και C=2

Statistic Measure	Naive Bayes	Random Forest	SVM	KNN
Accuracy	0.895	0.945	0.961	0.957
Precision	0.886	0.942	0.957	0.953
Recall	0.888	0.939	0.957	0.954
F-Measure	0.885	0.940	0.957	0.954
AUC				

Beat the Benchmark

Τα αρχικά μας αποτελέσματα ήταν αυτά με καλύτερο ταξινομητή τών KNN

Statistic Measure	Naive Bayes	Random Forest	SVM	KNN
Accuracy	0.892	0.934	0.935	0.941
Precision	0.879	0.929	0.932	0.936
Recall	0.891	0.928	0.927	0.936
F-Measure	0.882	0.928	0.929	0.936
AUC				

Βελτίωση 1 αλλαγή ταξινομητή

Δοκιμάζοντας τον SVM με τις παραμέτρους kernel(rbf,linear),c,gamma παρατήρησα ότι έχει καλύτερα αποτελέσματα από τον KNN. Υστερα από αρκετές δοκιμές με τις παραμέτρους παρατήρησα ότι για linear kernel και c=2 έχω τις καλύτερες τιμές γενικά. Καθώς το accuracy μας με cross validation 10 fold έβγαινε σχεδόν 0.967

Παρατήρηση : Είδαμε από το clustering ότι τα δεδομένα μας είναι αρκετά γραμμικά διαχωρίσιμα επομένως θεωρούμε πως για αυτό το λόγο ο svm δίνει καλύτερα αποτελέσματα με linear kernel.

Βελτίωση 2 preprocess LSI με διαφορετικο αριθμό components

Κατά την διάρκεια της εργασίας παρατηρήσαμε ότι η προεπεξεργασία των δεδομένων παίζει πολύ σημαντικό ρόλο στην απόδοση του ταξινομητή μας καθώς αρχικά χρησιμοποιήσαμε count vectorizer και είχαμε σχετικά κακό accuracy

Στην συνέχεια χρησιμοποιήσαμε tfid vectorizer με πολύ καλύτερα αποτέλεσματα.

Αυτό που έκανε όμως μεγάλη διαφορά στην απόδοση μας έιναι όταν κάναμε μετρήσεις στον ταξινομητή μας αλλάζοντας τον αριθμό διάστασης του LSI και παρακάτω παρουσιάζονται οι μετρήσεις μας

Latent Semantic Indexing (LSI) με διαφορετικό αριθμό components.

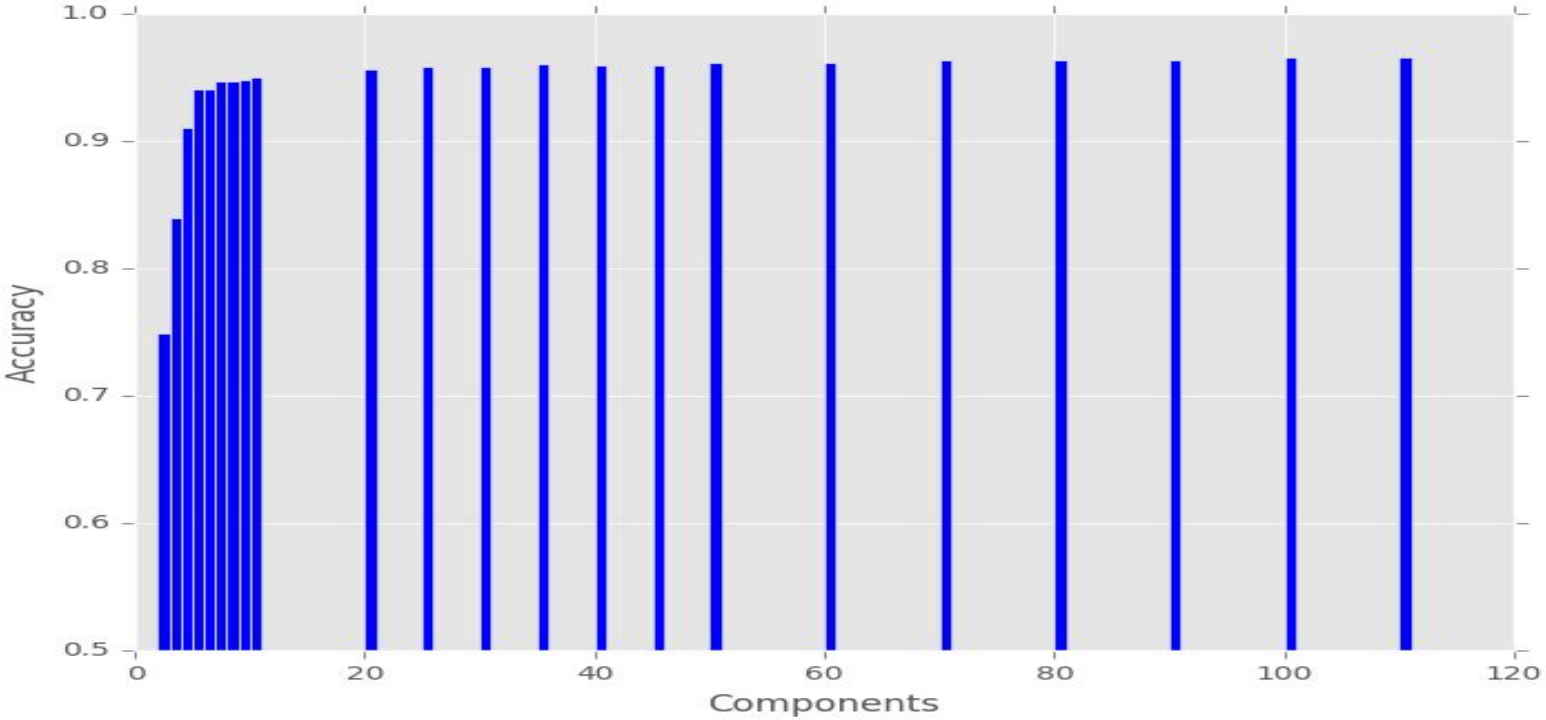
Για να παρατηρήσουμε το πώς επηρρεάζονται οι μετρήσεις μας ανάλογα με τον αριθμό components που χρησιμοποιούμε θα κρατήσουμε σταθερό τον ταξινητή μας και θα μεταβάλλουμε τον αριθμό τως components.

Ταξινομητής: Support Vector Machine (SVM) με linear kernel C=2

Μετρική : Accuracy

Components: Απο 1..120

Latent Semantic Indexing (LSI) με διαφορετικό αριθμό components.



Latent Semantic Indexing (LSI) με διαφορετικό αριθμό components.

Παρατηρούμε πως για μικρές διαστάσεις ο ταξινομητής δίνει πολύ κακά αποτελέσματα αλλά καθώς αυξάνουμε γίνεται όλο και καλύτερος.

Στις 100 διαστάσεις βλέπουμε ότι σταθεροποιείται γύρω στο 0.965 αλλά συνεχίζει να βελτιώνετε μεχρι και τις 120 διαστάσεις στα 0.9657.

Οι καλύτερες τιμές που πέτυχα ήταν με διαστασεις 300,400 με απόδοση 0.9676 και 0.9677 αντίστοιχα που συμφωνούν με την wikipedia https://en.wikipedia.org/wiki/Latent_semantic_analysis#Latent_semantic_indexing

Στην παράγραφο Challenges to LSI

Βελτίωση 3 Αλλαγή βάρους των λέξεων του τίτλου

Δοκίμασα διαφορετικά βάρη για τις λέξεις του τίτλου πολλαπλασιάζοντας κάθε φορά με διαφορετικό αριθμό το vector των λέξεων του τίτλου και παρατήρησα τα εξης:

Για μεγάλα βάρη(5,100,1000) έπαιρνα χειρότερα αποτελέσματα και αυτό είναι λογικό καθώς σημαίνει ότι πλέον αγνοούσα τις λέξεις του κειμένου.

Για μικρά βάρη(2,3) έπαιρνα σχεδόν ίδιες τιμές

Για βάρος 1.1 πήρα μια πολύ μικρή βελτίωση.