ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή εργασία

Τεχνικές Μηχανικής Μάθησης για ανίχνευση ψευδών ειδήσεων στα κοινωνικά δίκτυα

Βασίλειος Κυριακόπουλος
2022201800103

Σπυρίδων Μάης
2022201300176

Επιβλέπουσα:

Παρασκευή Ραυτοπούλου
ΕΔΙΠ

Τρίπολη, Απρίλιος, 2022

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

Περίληψη

Στην πτυχιακή εργασία μελετήθηκε η εφαρμογή αλγόριθμων Μηχανικής Μάθησης για τον εντοπισμό ψευδών ειδήσεων στο Twitter σχετικά με την πανδημία του Covid-19. Για τον σκοπό αυτό, συλλέχθηκαν συλλογές δεδομένων από το Twitter, και στην συνέχεια εκτελέστηκαν 10 διαφορετικοί αλγόριθμοι κατηγοριοποίησης, αλλά και διαφορετικές μέθοδοι μετατροπής των κειμένων σε διανύσματα. Συγκεκριμένα, εφαρμόστηκαν οι μέθοδοι Count Vectorizer, Tf-Idf Vectorizer με και χωρίς την εφαρμογή Stemming, με διάφορες επιλογές τιμών N-Gram, και έπειτα εφαρμόστηκαν αλγόριθμοι επιβλεπόμενης Μηχανικής Μάθησης. Για την εκτέλεση όλων των δοκιμών υλοποιήθηκαν προγράμματα στην γλώσσα Python, και χρησιμοποιήθηκαν βιβλιοθήκες όπως οι Numpy, Scikit-Learn και Pandas. Για τον κάθε συνδυασμό αλγόριθμου – διανυσματοποίησης, καταγράφηκαν οι μετρικές των αποτελεσμάτων πρόβλεψης, όπως το F1-Score, Accuracy, confusion matrix και εν τέλει επιλέχθηκαν οι συνδυασμοί αλγόριθμων - διανυσματοποιήσεων που έφεραν τα βέλτιστα αποτελέσματα.

Abstract

In the present thesis we have studied the application of Machine Learning algorithms for fake news detection on Twitter about Covid-19 pandemic. For this purpose, we have collected data from Twitter and applied 10 different machine learning algorithms, as well as different methods of text vectorization. Specifically, we have applied Bag of Words, Tf-Idf vectorization methods, with and without the use of Stemming, with different N-Gram values. The resulting vectors were then processed by supervised Machine Learning Algorithms. For all the experiments, we implemented Python programs and used libraries such as ScikitLearn, Numpy and Pandas. For each algorithm – vectorization combination, we have recorded a number of metrics, such as F1-Score, Accuracy and confusion matrix and in the end we chose the combinations of algorithms - vectorizations that brought the best results.

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός

Σκοπός της παρούσας πτυχιακής εργασίας είναι να εντοπίσει τις ψευδείς ειδήσεις σχετικά με τον Covid-19 σε μια συλλογή δεδομένων από το Twitter. Ο εντοπισμός των ψευδών ειδήσεων θα γίνει με την χρήση τεχνικών Μηχανικής Μάθησης. Για τον σκοπό αυτό, βασιστήκαμε στο scikit-learn [1], μια βιβλιοθήκη της γλώσσας προγραμματισμού Python, που χρησιμοποιείται ευρέως για Μηχανική Μάθηση και στατιστική μοντελοποίηση. Η συγκεκριμένη βιβλιοθήκη διαθέτει διάφορους classification, regression και clustering αλγόριθμους συμπεριλαμβανομένων support-vector machines, random forests, gradient boosting, k-means και έχει σχεδιαστεί για να λειτουργεί με τις βιβλιοθήκες Pandas και Numpy της Python. Επίσης το scikit-learn προσφέρει μια ευκολία στον χρήστη για την εφαρμογή των αλγορίθμων σε μια συλλογή δεδομένων κάτι το οποίο μας βοήθησε στα πλαίσια της παρούσας πτυχιακής εργασίας. Τελικά καταφέραμε να μελετήσουμε δεδομένα Ελλήνων χρηστών του Twitter από την αρχή της πανδημίας (Μάρτιος 2020) έως τον Δεκέμβριο του 2021 και δεδομένα σε αγγλική γλώσσα από διάφορα κοινωνικά δίκτυα, όπως το Twitter, από τις 16 Οκτωβρίου 2020 έως τις 16 Δεκέμβριου 2020.

1.2 Ψευδείς Ειδήσεις

Η ελευθερία στην έκφραση και πρόσβαση της πληροφορίας σε συνδυασμό με την ανάπτυξη της τεχνολογίας και των μέσων επικοινωνίας και ενημέρωσης είχε ως φυσικό επακόλουθο τη δημιουργία πολλών ψευδών ειδήσεων στο χώρο του διαδικτύου. Η κατασκευή αυτών των ψευδών ειδήσεων έχουν ως κύριο στόχο την παραπλάνηση, την παραπληροφόρηση, την πρόκληση φόβου και άλλων αισθημάτων, με απώτερο σκοπό είτε τη χειραγώγηση της κοινής γνώμης είτε την επίτευξη οικονομικού κέρδους. Αν και το φαινόμενο των ψευδών ειδήσεων δεν είναι κάτι νέο, τα τελευταία χρόνια απασχολεί έντονα ερευνητές, διεθνείς οργανισμούς και κυβερνήσεις, καθώς τα μέσα κοινωνικής δικτύωσης έχουν αλλάξει τον τρόπο διανομής των πληροφοριών, αυξάνοντας την ταχύτητα διάδοσης και εξάπλωσης των ψευδών ειδήσεων.

Σε αυτό το σημείο είναι σημαντικό να αναφερθεί ότι λόγω των μέσων κοινωνικής

[1] https://scikit-learn.org/stable/

δικτύωσης οι αναγνώστες μπορούν πιο εύκολα να κοινοποιήσουν ό,τι διαβάζουν στο διαδίκτυο χωρίς να ξέρουν εάν αυτό είναι αληθινό γεγονός ή ψευδής είδηση. Το πρόβλημα στην αναγνώριση των ψευδών ειδήσεων προκύπτει όταν τις κοινοποιούν ειδησεογραφικοί οργανισμοί, οι οποίοι θεωρούνται ότι αναρτούν έγκυρες και αληθείς ειδήσεις.

1.3 Twitter και η πανδημία Covid-19

Τα τελευταία χρόνια ο πλανήτης δοκιμάζεται από την πανδημία του Covid-19,ενός νέου πολύ μεταδοτικού ιού, ο οποίος εμφανίστηκε πρώτη φορά στην επαρχία Wuhan της Κίνας το χειμώνα του 2019. Μαζί με την εξάπλωση της πανδημίας δημιουργήθηκε ταυτόχρονα ένα τεράστιο κύμα ψευδών ειδήσεων σχετικό με τον ιό, την αντιμετώπιση του και την προέλευση του. Οι ψευδείς ειδήσεις σε μια πανδημία μπορούν να θέσουν τη δημόσια υγεία σε κίνδυνο, ιδιαίτερα εάν προτρέπουν τους χρήστες να μην ακολουθούν τα μέτρα προστασίας, καλλιεργούν τις διακρίσεις ή ενθαρρύνουν τις πράξεις βίας και υποστηρίζουν τη χρήση μη εγκεκριμένων και δοκιμασμένων θεραπειών. Με τον τρόπο αυτό σαμποτάρουν την αποτελεσματικότητα των μέτρων προστασίας, οδηγούν το κοινό σε σύγχυση και διαταράσσουν την κοινωνική συνοχή.

Καθώς η επιδημία έκανε την επέλασή της, εκατομμύρια άνθρωποι βρέθηκαν αποκλεισμένοι στα σπίτια τους, αντιμέτωποι με τον ιό. Οι άνευ προηγουμένου συνθήκες όπως είναι αναμενόμενο δημιούργησαν αυξημένες ανάγκες για πληροφόρηση. Δεδομένου ότι ένα μεγάλο ποσοστό των πολιτών χρησιμοποιούν το διαδίκτυο καθημερινά καθώς και τα μέσα κοινωνικής δικτύωσης είναι λογικό ο κόσμος να στράφηκε σε αυτά τα μέσα για την ενημέρωσή του. Το διαδίκτυο και οι πλατφόρμες κοινωνικής δικτύωσης αποτέλεσαν το εύφορο έδαφος για κάθε είδους θεωρίες συνομωσίας, λανθασμένες συμβουλές θεραπείας αλλά και για οργανωμένες προσπάθειες παραπληροφόρησης και προπαγάνδας.

Το Twitter [2] είναι μια από τις πιο διάσημες πλατφόρμες μέσων κοινωνικής δικτύωσης που επιτρέπει στους χρήστες να δημοσιεύουν δεδομένα περίπου 140 χαρακτήρων που αναφέρονται ως ”tweet”. Το Twitter δίνει στους χρήστες την ευκαιρία να μοιραστούν τα μηνύματά τους σχετικά με οτιδήποτε χαρακτηρίζεται ως γνήσιο, συμπεριλαμβανομένων ειδήσεων, πολιτικών ζητημάτων, προσωπικών απόψεων και ούτω καθεξής. Όπως επισημαίνεται από το Twitter, το Twitter έχει 396 εκατομμύρια χρήστες και 206 εκατομμύρια καθημερινά ενεργούς χρήστες [29]. Λόγω του μεγάλου όγκου χρηστών και του τρόπου λειτουργίας του Twitter οι ψευδείς ειδήσεις σχετικά με την πανδημία αυξάνονται συνεχώς με την πάροδο του χρόνου. Ο γενικός διευθυντής του Π.Ο.Υ Dr.Tedros Adhanom Ghebreyesus, κατά τη διάρκεια της ομιλίας του στη Διάσκεψη Ασφαλείας του Μονάχου, επεσήμανε ότι εκτός από τη μάχη κατά της επιδημίας του Covid-19, υπάρχουν και άλλες παράπλευρες μάχες που πρέπει να δοθούν: αυτές ενάντια στη λανθασμένη πληροφόρηση και στη ψευδή πληροφόρηση. Δήλωσε χαρακτηριστικά «…δεν παλεύουμε απλά ενάντια σε μια επιδημία, παλεύουμε ενάντια σε μια επιδημία πληροφόρησης. Οι ψευδείς ειδήσεις διαδίδονται ευκολότερα και γρηγορότερα από τον ιό, και είναι εξίσου επικίνδυνες.» [30]

[2] https://twitter.com/

1.4 Μηχανική Μάθηση για τον εντοπισμό ψευδών ειδήσεων

Η Μηχανική Μάθηση είναι υπό-πεδίο της επιστήμης των υπολογιστών που αναπτύχθηκε από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη [31]. Η Μηχανική Μάθηση διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Τέτοιοι αλγόριθμοι λειτουργούν κατασκευάζοντας μοντέλα από πειραματικά δεδομένα, προκειμένου να κάνουν προβλέψεις βασιζόμενες στα δεδομένα ή να εξάγουν αποφάσεις που εκφράζονται ως το αποτέλεσμα [31]. Η Μηχανική Μάθηση είναι στενά συνδεδεμένη και συχνά συγχέεται με την υπολογιστική στατιστική, ένας κλάδος, που επίσης επικεντρώνεται στην πρόβλεψη μέσω της χρήσης των υπολογιστών. Έχει ισχυρούς δεσμούς με την μαθηματική βελτιστοποίηση, η οποία της παρέχει μεθόδους, την θεωρία και τομείς εφαρμογής. Η Μηχανική Μάθηση εφαρμόζεται σε μια σειρά από υπολογιστικές εργασίες, όπου τόσο ο σχεδιασμός όσο και ο ρητός προγραμματισμός των αλγορίθμων είναι ανέφικτος. Παραδείγματα εφαρμογών αποτελούν τα φίλτρα spam (spam filtering), η οπτική αναγνώριση χαρακτήρων (OCR), οι μηχανές αναζήτησης και η υπολογιστική όραση [32]. Η Μηχανική Μάθηση μερικές φορές συγχέεται με την εξόρυξη δεδομένων, όπου η τελευταία επικεντρώνεται περισσότερο στην εξερευνητική ανάλυση των δεδομένων, γνωστή και ως μη επιτηρούμενη μάθηση.

Ο Tom M. Mitchell πρότεινε έναν πιο επίσημο ορισμό που χρησιμοποιείται ευρέως: «Ένα πρόγραμμα υπολογιστή λέγεται ότι μαθαίνει από εμπειρία E ως προς μια κλάση εργασιών T και ένα μέτρο επίδοσης P, αν η επίδοσή του σε εργασίες της κλάσης T, όπως αποτιμάται από το μέτρο P, βελτιώνεται με την εμπειρία E» [35]. Αυτός ο ορισμός είναι σημαντικός για τον καθορισμό της Μηχανικής Μάθησης σε βασικό λειτουργικό πλαίσιο παρά με γνωστικούς όρους, ακολουθώντας έτσι την πρόταση του Alan Turing στην εργασία του «Υπολογιστικές μηχανές και Νοημοσύνη»[33], ότι το ερώτημα αν μπορούν οι μηχανές να σκεφτούν, μπορεί να αντικατασταθεί με το ερώτημα αν μπορούν οι μηχανές να κάνουν αυτό που εμείς (ως σκεπτόμενες οντότητες) μπορούμε να κάνουμε.

Η επιστήμη υπολογιστών και η Μηχανική Μάθηση, αποδεικνύεται ότι είναι πολύ πιθανόν κατάλληλες για τον εντοπισμό του φαινομένου των ψευδών ειδήσεων. Χρησιμοποιώντας κλασσικούς αλγορίθμους Μηχανικής Μάθησης σε συνδυασμό με τεχνικές Natural Language Processing (NLP) [34] μπορούμε να προσεγγίσουμε σε ικανοποιητικό βαθμό την ανίχνευση των ψευδών ειδήσεων ακόμα και σε ένα μικρό δείγμα δεδομένων.

1.5 Σχετικές Υλοποιήσεις

Υπάρχουν αρκετές υλοποιήσεις του προβλήματος της ανίχνευσης ψευδών ειδήσεων που παρουσιάζουν ενδιαφέρον έχοντας διαφορετικές προσεγγίσεις στις μεθόδους που προτείνουν.

Ο De Wang πρότεινε ένα framework [24] για την ανίχνευση spam μηνυμάτων. Στην υλοποίηση των Benjamin Markines, Ciro Cattuto και Filippo Menczer [25], οι συγγραφείς εντοπίζουν και αναλύουν έξι διαφορετικούς τρόπους για τον εντοπισμό ανεπιθύμητων μηνυμάτων. Χρησιμοποιούν αυτές τις δυνατότητες για να διακρίνουν τους spammers από τους γνήσιους χρήστες. Ο Xueying Zhang και η ομάδα του [26] προτείνουν έναν επανα-

στατικό μοντέλο Μηχανικής Μάθησης για τον εντοπισμό spammers. Η προσομοίωση αυτού του αλγορίθμου πραγματοποιείται στο περιβάλλον του MATLAB και προσφέρει 99,1 % αποτελεσματικότητα για τον εντοπισμό των spammers και 99,9 % των non-spammers. Οι Hailu Xu,Weiqing Sun, Ahmad Javaid [27] κάνουν μια προσέγγιση αναγνώρισης ανεπιθύμητων μηνυμάτων χρησιμοποιώντας σύνολα δεδομένων από τις πλατφόρμες Facebook και Twitter. Χρησιμοποιούν σχεδόν κοινά χαρακτηριστικά μεταξύ δύο συνόλων δεδομένων για τον εντοπισμό ανεπιθύμητων μηνυμάτων. Σε αυτή τη μέθοδο συνδυάζουν spam δεδομένα από το Twitter και από το Facebook για εκπαίδευση και ο αλγόριθμος random forest δίνει τα καλύτερα αποτελέσματα σε σχέση με τους υπόλοιπους αλγορίθμους Μηχανικής Μάθησης που χρησιμοποιούνται στην εργασία αυτή. Άλλη μια υλοποίηση αποτελεί το GROVER [28]. Το σύστημα αυτό, έχει ως στόχο την αναγνώριση ψευδών ειδήσεων, οι οποίες παράχθηκαν από Νευρωνικά Δίκτυα. Για το σκοπό αυτό, το GROVER είναι ένα σύστημα που παράγει ψευδείς ειδήσεις με την χρήση Deep Neural Networks, αλλά μπορεί επίσης να αναγνωρίζει άρθρα που συντάχθηκαν με ανάλογο τρόπο.

Τέλος να σημειώσουμε ότι υπάρχει πληθώρα σχετικών δημοσιεύσεων και υλοποιήσεων και αρκετό ερευνητικό ενδιαφέρον, και αυτά που αναφέραμε αποτελούν ένα μικρό δείγμα των εξελίξεων στον τομέα.

1.6 Συνεισφορά

Η συνεισφορά της εργασίας μας μας συνοψίζεται στα εξής:

1. Συγκεντρώσαμε με scraping μηχανισμούς 2MB δεδομένων που αντιστοιχούν σε παραπάνω από 1200 Tweets Ελλήνων χρηστών και αφορούν το διάστημα από την αρχή της πανδημίας (Μάρτιος 2020) μέχρι και τον Δεκέμβρη του 2021.

2. Έχουμε αξιολογήσει διαφορετικά μοντέλα για την κατασκευή των διανυσμάτων των δεδομένων, όπως επίσης και διαφορετικές τεχνικές στην προεπεξεργασία των tweets. Όλες οι επιλογές έχουν αξιολογηθεί πειραματικά με τη χρήση γνωστών μετρικών, όπως η ακρίβεια (accuracy) και το F1-measure. Τα αποτελέσματα της πειραματικής αξιολόγησης παρουσιάζονται αναλυτικά στην παρούσα εργασία.

3. Έχουμε δοκιμάσει μια σειρά από τεχνικές μηχανικής μάθησης για την ταξινόμηση των tweets σε αληθείς και ψευδείς ειδήσεις. Όλοι οι αλγόριθμοι έχουν δοκιμαστεί σε δυο διαφορετικά σύνολα δεδομένων, σε ένα που περιλαμβάνει ελληνικά κι σε ένα με αγγλόφωνα tweets που αφορούν τον Covid-19.

1.7 Οργάνωση πτυχιακής

Η πτυχιακή είναι χωρισμένη σε έξι κεφάλαια. Σε αυτό το κεφάλαιο παρουσιάζουμε την δομή και την οργάνωση της πτυχιακής. Στο δεύτερο κεφάλαιο παρουσιάζονται οι αλγόριθμοι Μηχανικής Μάθησης που χρησιμοποιήθηκαν και ο τρόπος λειτουργίας τους, καθώς επίσης το μοντέλο επεξεργασίας φυσικής γλώσσας (NLP) και οι μετρικές αξιολόγησης των αλγορίθμων. Στο τρίτο κεφάλαιο θα αναλύσουμε τον τρόπο που μαζέψαμε

από το Twitter τα δεδομένα καθώς επίσης και τα βήματα της προετοιμασίας που χρειάστηκε να εφαρμόσουμε πάνω σε αυτά έτσι ώστε να είναι κατάλληλα ως είσοδος για τους αλγορίθμους Μηχανικής Μάθησης. Στο τέταρτο κεφάλαιο βρίσκονται τα αποτελέσματα και τα συμπεράσματα από την εκτέλεση των αλγορίθμων. Στο πέμπτο κεφάλαιο βρίσκεται ο επίλογος της πτυχιακής και κάποιες ιδέες για μελλοντικές υλοποιήσεις. Τέλος στο έκτο κεφάλαιο παραθέτουμε την βιβλιογραφία που χρησιμοποιήθηκε για την βοήθεια της συγγραφής της παρούσας πτυχιακής.

Κεφάλαιο 2

Αλγόριθμοι Μηχανικής Μάθησης

Σε αυτό το κεφάλαιο παρουσιάζεται μια επισκόπηση της επιστήμης της Μηχανικής Μάθησης, και εν συνεχεία εξηγούμε τους διαθέσιμους αλγόριθμους κατηγοριοποίησης.

2.1 Εισαγωγικές έννοιες

Η μάθηση θα μπορούσε να χαρακτηριστεί ως η ικανότητα μετατροπής προηγούμενης εμπειρίας σε γνώση. Με αντίστοιχο τρόπο επιδιώκεται η λύση προβλημάτων στον τομέα της Μηχανικής Μάθησης. Ο Tom Mitchel από το Carnegie Mellon University ορίζει την Μηχανική Μάθηση ως την μελέτη των αλγορίθμων που επιτρέπουν στα προγράμματα να βελτιωθούν αυτόματα μέσω της εμπειρίας [35]. Ως είσοδος σε ένα σύστημα Μηχανικής Μάθησης, δίνεται ένα σύνολο στοιχείων με το αποτέλεσμα που φέρουν, όπως π.χ. ένα σύνολο από Tweets. Το σύνολο των δεδομένων αυτών, ονομάζεται Δεδομένα Εκπαίδευσης (Training Dataset). Το σύστημα Μηχανικής Μάθησης, παρατηρεί τα δεδομένα αυτά, και δημιουργεί μια συνάρτηση, η οποία επιδιώκει να λαμβάνει ως είσοδο τα δεδομένα και να εξάγει το αποτέλεσμα της ετικέτας με την μέγιστη δυνατή ακρίβεια. Έτσι το σύστημα παίρνοντας ως είσοδο νέα, άγνωστα δεδομένα, με άγνωστη ετικέτα, μπορεί να προβλέψει με επιτυχία την ετικέτα.

Η λειτουργία των συστημάτων Μηχανικής Μάθησης εξαρτάται από πολλούς παράγοντες. Ένας σημαντικός παράγοντας, είναι η ποιότητα των δεδομένων. Είναι κομβικό τα δεδομένα να είναι αξιόπιστα χωρίς ελλείψεις, γιατί σε διαφορετική περίπτωση η διαδικασία της μάθησης θα είναι αναξιόπιστη και το αποτέλεσμα δεν θα είναι το επιθυμητό. Άλλος σημαντικός παράγοντας, είναι ο όγκος των δεδομένων αλλά και η κατάλληλη επιλογή των παραμέτρων που θα δοθούν ως δεδομένα εκπαίδευσης. Η επιστήμη της Μηχανικής Μάθησης έχει ως στόχο να επιλύσει προβλήματα για τα οποία είναι πολύπλοκο να προγραμματιστεί ένα σύστημα να τα επιλύει, όπως για παράδειγμα η αναγνώριση εικόνας, αλλά και προβλήματα τα οποία είναι αδύνατον να επιλυθούν από ένα άνθρωπο, όπως για παράδειγμα η ανάκτηση πληροφορίας από μεγάλους όγκους δεδομένων, όπως τα δεδομένα των μηχανών αναζήτησης διαδικτύου. Με τη πάροδο του χρόνου η Μηχανική Μάθηση έχει εφαρμογές σε ποικίλα επιστημονικά πεδία. Μερικά παραδείγματα είναι η αναγνώριση ομιλίας και γραφικού χαρακτήρα, η ανάκτηση πληροφορίας, η βίο-πληροφορική και πολλά άλλα. Επίσης, η Μηχανική Μάθηση καθιστά εφικτά πολλά τεχνολογικά επιτεύγματα, όπως η μετακίνηση ρομπότ, ανάλυση DNA και οι προσπάθειες για την εξερεύνηση

του διαστήματος[2].

Η Μηχανική Μάθηση χωρίζεται σε τρεις βασικές κατηγορίες:

1. Στην επιβλεπόμενη μάθηση, τα δεδομένα εκπαίδευσης του αλγόριθμου περιέχουν και την επιθυμητή έξοδο, έτσι το σύστημα παράγει ένα μοντέλο το οποίο μπορεί να χρησιμοποιηθεί για να χαρακτηρίσει νέα παραδείγματα. Είναι η διαδικασία δηλαδή όπου ο αλγόριθμος κατασκευάζει μια συνάρτηση που απεικονίζει δεδομένες εισόδους (σύνολο εκπαίδευσης) σε γνωστές επιθυμητές εξόδους, με απώτερο στόχο τη γενίκευση της συνάρτησης αυτής και για εισόδους με άγνωστη έξοδο. Χρησιμοποιείται σε προβλήματα πρόβλεψης (Prediction), Ταξινόμησης (Classification) και Παλινδρόμησης (Regression) [3].

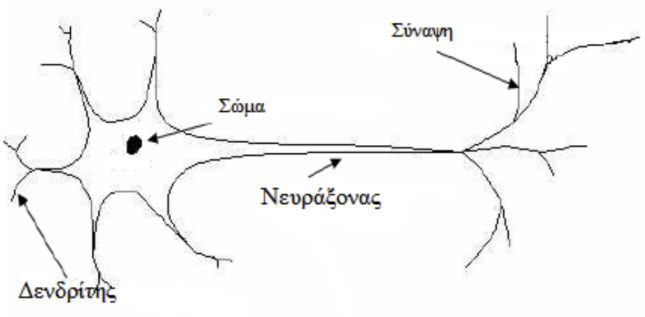
2. Στην μη επιβλεπόμενη μάθηση, τα δεδομένα εκπαίδευσης δεν περιέχουν την επιθυμητή έξοδο. Σκοπός ενός τέτοιου συστήματος, είναι η δημιουργία ενός μοντέλου, το οποίο δύναται να εντοπίσει κάποιο κρυμμένο μοτίβο στα δεδομένα [4].

3. Ενισχυτική μάθηση (Reinforcement learning) είναι η διαδικασία κατά την οποία ο αλγόριθμος μαθαίνει μια στρατηγική ενεργειών αλληλοεπιδρώντας άμεσα με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτευχθεί ένας συγκεκριμένος στόχος. Χρησιμοποιείται κυρίως σε προβλήματα σχεδιασμού (planning), όπως για παράδειγμα ο έλεγχος κίνησης ρομπότ ή για να μάθει να παίζει ένα παιχνίδι εναντίον κάποιου αντιπάλου. Επίσης, η ενισχυτική μάθηση χρησιμοποιείται και για την υποβοήθηση της οδήγησης ή για την πλήρως αυτοματοποιημένη οδήγηση σε έξυπνα οχήματα [3].

2.1.1 Τεχνητοί νευρώνες - μια σύντομη ματιά στην ιστορία της Μηχανικής Μάθησης

Πριν συζητήσουμε αλγόριθμους Μηχανικής Μάθησης, ας κάνουμε μια σύντομη περιήγηση στις απαρχές της Μηχανικής Μάθησης. Προσπαθώντας να καταλάβουμε πώς ο βιολογικός εγκέφαλος λειτουργεί, για να σχεδιάσει την τεχνητή νοημοσύνη (AI), ο Warren McCulloch και ο Walter Pitts δημοσίευσαν την πρώτη ιδέα ενός απλοποιημένου εγκεφαλικού κυττάρου, ο λεγόμενος νευρώνας McCulloch-Pitts (MCP), το 1943 [36]. Οι βιολογικοί νευρώνες Σχήμα: 2.1 είναι διασυνδεδεμένα νευρικά κύτταρα στον εγκέφαλο που εμπλέκονται στην επεξεργασία και τη μετάδοση χημικών και ηλεκτρικών σημάτων.

Οι McCulloch και Pitts περιέγραψαν ένα τέτοιο νευρικό κύτταρο ως μια απλή λογική πύλη με δυαδικές εξόδους, πολλαπλά σήματα φτάνουν στους δενδρίτες και στη συνέχεια ενσωματώνονται στο κυτταρικό σώμα και εάν το συσσωρευμένο σήμα υπερβαίνει ένα ορισμένο όριο, ένα σήμα εξόδου δημιουργείται που θα μεταδοθεί από τον νευράξονα. Μόλις λίγα χρόνια αργότερα, ο Frank Rosenblatt δημοσίευσε την πρώτη ιδέα του αλγορίθμου Μηχανικής Μάθησης perceptron που βασίζεται στο μοντέλο νευρώνων MCP [37]. Με αυτό το μοντέλο του perceptron, ο Rosenblatt πρότεινε έναν αλγόριθμο που θα μάθαινε αυτόματα τους βέλτιστους συντελεστές βάρους που στη συνέχεια θα πολλαπλασιάζονταν με τα χαρακτηριστικά εισόδου προκειμένου να ληφθεί η απόφαση για το αν ένας νευρώνας πυροδοτεί (μεταδίδει σήμα) ή όχι. Στο πλαίσιο της εποπτευόμενης μάθησης και ταξινόμησης, ένας τέτοιος αλγόριθμος θα μπορούσε να χρησιμοποιείται για να προβλέψει εάν



Σχήμα 2.1: Νευρώνας McCulloch-Pitts (MCP)

ένα νέο σημείο δεδομένων ανήκει στη μία κατηγορία ή σε μια άλλη [6][7][8].

2.2 Αλγόριθμοι Κατηγοριοποίησης

Η κατηγοριοποίηση είναι εργασία επιβλεπόμενης μάθησης, που στόχο έχει την ανακάλυψη της σχέσης ανάμεσα σε ένα γνώρισμα στόχο με ονομαστικές τιμές και σε ένα σύνολο άλλων γνωρισμάτων. Η διαδικασία της κατηγοριοποίησης περιλαμβάνει τρία στάδια. Στο πρώτο στάδιο ο αλγόριθμος επεξεργάζεται τα δεδομένα του συνόλου εκπαίδευσης και κατασκευάζει ένα μοντέλο. Στο δεύτερο στάδιο ελέγχεται η ικανότητα του μοντέλου να προβλέπει την κλάση άγνωστων παρατηρήσεων. Εάν η επίδοση του μοντέλου κριθεί ικανοποιητική, τότε ακολουθεί το τρίτο στάδιο, το οποίο συνίσταται στη χρήση του μοντέλου για τη διατύπωση προβλέψεων. Κατά την εκπαίδευση πρέπει να αποφευχθεί η υπερπροσαρμογή του μοντέλου, η απομνημόνευση δηλαδή του συγκεκριμένου συνόλου εκπαίδευσης. Αποτέλεσμα της υπερπροσαρμογής είναι η πτώση της επίδοσης έναντι άγνωστων παρατηρήσεων. Κριτήρια για την αξιολόγηση των μεθόδων κατηγοριοποίησης είναι η ακρίβεια πρόβλεψης, η ταχύτητα, η ερμηνευσιμότητα, η επεκτασιμότητα και η ανθεκτικότητα. [5]

2.2.1 Γραμμικοί αλγόριθμοι κατηγοριοποίησης

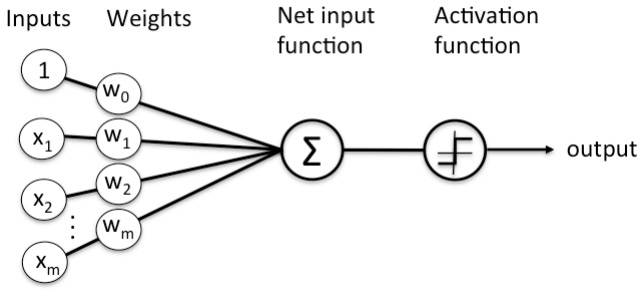
Στόχος των γραμμικών αλγόριθμων κατηγοριοποίησης, είναι ο κατάλληλος γραμμικός συνδυασμός των χαρακτηριστικών του διανύσματος εισόδου, έτσι ώστε να γίνει σωστή πρόβλεψη της κατηγορίας που ανήκει το διάνυσμα. Ως παράδειγμα δυαδικής κατηγοριοποίησης, δεδομένου ότι τα διανύσματα εισόδου είναι της μορφής $x_i = [x_1, x_2, ..., x_n]$, σκοπός είναι η δημιουργία ενός διανύσματος $w = [w_1, w_2, ..., w_n]$ έτσι ώστε με τον πολλαπλασιασμό διανυσμάτων $wx_i = h(x_i)$ το βαθμωτό μέγεθος που προκύπτει να κατηγοριοποιείται με ελάχιστη απόσταση από την πραγματική κατηγορία που ανήκει. Η κατηγορία πρόβλεψης $\hat{y}$ του διανύσματος καθορίζεται από την τιμή της συνάρτησης $h(x_i)$, και της τιμής που δίνεται στο κατώφλι απόφασης T, και ορίζεται ως:

$$\hat{y} = \begin{cases} 1 & \text{if } wx_i \geq T, \\ 0 & \text{otherwise.} \end{cases}$$

Κάποιοι από τους αλγόριθμους που ανήκουν στην κατηγορία αυτή, και αποτελούν μέρος της υλοποίησης της εργασίας αυτής είναι:
• Perceptron [6]
• Logistic Regression [6]
• Ridge Classifier [6]

Ο ορισμός ενός τεχνητού νευρώνα και ο Perceptron classifier

Η ιδέα πίσω από τους τεχνητούς νευρώνες μπορεί να εφαρμοστεί σε μια διαδικασία δυαδικής ταξινόμησης όπου αναφερόμαστε στις δύο τάξεις μας ως 1 (θετική κλάση) και –1 (αρνητική κλάση).



Σχήμα 2.2: Παράδειγμα Perceptron Classifier

Στη συνέχεια μπορούμε να ορίσουμε μια συνάρτηση απόφασης $\varphi(z)$ που παίρνει έναν γραμμικό συνδυασμό ορισμένων τιμών εισόδου x και ένα αντίστοιχο διάνυσμα βάρους w, όπου $z = w^T x = w_1 x_1 + w_2 x_2 + ... + w_m x_m$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_m \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_m \end{bmatrix}$$

Αν το z για μια συγκεκριμένη είσοδο $x^{(i)}$ είναι μεγαλύτερο από το κατώφλι θ τότε το x ανήκει στην κλάση 1 αλλιώς στην -1. Στον αλγόριθμο perceptron, η συνάρτηση απόφασης, $\varphi(z)$, είναι μια παραλλαγή της συνάρτησης βήματος:

$$\varphi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise.} \end{cases}$$

Μπορούμε να φέρουμε το θ στην αριστερή πλευρά της εξίσωσης και ορίζουμε το $w_0 = -\theta$ και $x_0 = 1$, έτσι:

$$z = w^T x = w_0 x_0 + w_1 x_1 + ... + w_m x_m$$

Το βάρος, $w_0 = \theta$, ονομάζεται μονάδα προκατάληψης (bias unit). Ο αλγόριθμος Perceptron μπορεί να συνοψιστεί με τα ακόλουθα βήματα:

1. Αρχικοποιούμε τα βάρη του αλγορίθμου σε 0 ή μικρά τυχαία νούμερα
2. Για κάθε είσοδο x:
   (α) Υπολογίζουμε την έξοδο του αλγορίθμου $\hat{y}$
   (β) Ανανεώνουμε τις τιμές των βαρών με τον τύπο $w_j = w_j + \Delta w_j$.

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

Όπου η είναι ο ρυθμός εκμάθησης (συνήθως μια σταθερά μεταξύ 0 και 1), το $y^{(i)}$ είναι η κλάση της εισόδου ενώ $\hat{y}^{(i)}$ είναι η κλάση που προβλέπει ο αλγόριθμος. Στο Σχήμα 2.2 φαίνεται σχηματικά ένα παράδειγμα τέτοιου αλγορίθμου. Είναι σημαντικό να σημειωθεί ότι όλα τα βάρη τους διανύσματος ενημερώνονται ταυτόχρονα, που σημαίνει ότι δεν υπολογίζουμε εκ νέου την προβλεπόμενη κλάση προτού ενημερωθούν όλα τα βάρη μέσω του Δw_j[6].

Logistic regression classifier

Η λογιστική παλινδρόμηση είναι ένα μοντέλο ταξινόμησης που είναι πολύ εύκολο να εφαρμοστεί και αποδίδει πολύ καλά σε γραμμικά διαχωρίσιμες κλάσεις. Είναι από τους πιο ευρέως χρησιμοποιούμενους αλγόριθμους ταξινόμησης. Για να καταλάβουμε την ιδέα πίσω από την λογιστική παλινδρόμηση ως πιθανολογικό μοντέλο για δυαδική κατηγοριοποίηση, θεωρούμε την πιθανότητα p ότι μία είσοδος ανήκει σε μία κλάση. Έτσι ορίζουμε την συνάρτηση logit που είναι ίση με:

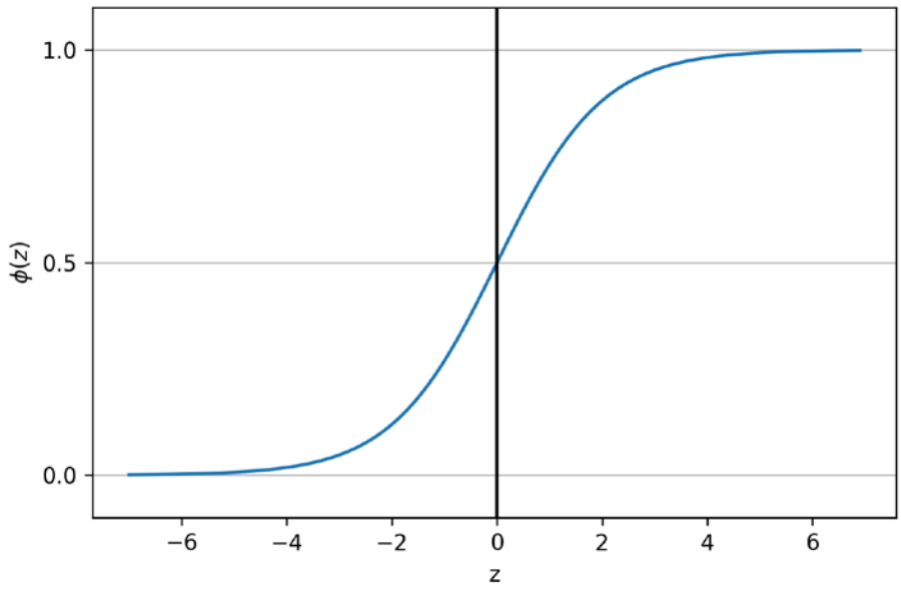
$$logit(p) = \log \frac{p}{(1-p)}$$

Σημειώνουμε ότι το log αναφέρεται στον φυσικό λογάριθμο, όπως είναι η κοινή σύμβαση στην επιστήμη των υπολογιστών. Η συνάρτηση logit λαμβάνει τιμές εισόδου στην περιοχή από 0 έως 1 και τα μετατρέπει σε τιμές σε όλο το εύρος πραγματικών αριθμών, τις οποίες μπορούμε να χρησιμοποιήσουμε για να εκφράσουμε την παρακάτω γραμμική σχέση:

$$logit(p(y=1|x)) = w_0x_0 + w_1x_1 + ... + w_mx_m = \sum_{i=0}^{m} w_ix_i = w^Tx$$

Εδώ το $logit(p(y=1|x)$ είναι η δεσμευμένη πιθανότητα ότι κάτι ανήκει στην κλάση 1 δεδομένου ότι η είσοδος ισούται με $x = x_1, x_2, ..., x_n$. Τώρα, μας ενδιαφέρει να προβλέψουμε την πιθανότητα ένα διάνυσμα x να ανήκει σε μια συγκεκριμένη κλάση, η οποία είναι η αντίστροφη μορφή της συνάρτησης logit. Ονομάζεται επίσης λογιστική σιγμοειδής συνάρτηση Σχήμα 2.3 :

$$\varphi(z) = \frac{1}{1+e^{-z}}$$



Σχήμα 2.3: Σιγμοειδής καμπύλη

Το z ισούται με τον γραμμικό συνδυασμό των βαρών w και της είσοδο x του αλγορίθμου, $z = w^T x = w_0 x_0 + w_1 x_1 + ... + w_m x_m$. Άρα αν το αποτέλεσμα της συνάρτησης είναι μεγαλύτερο ή ίσο με 0.5 το x ανήκει στην κλάση 1 αλλιώς στην κλάση 0. Έτσι εύκολα μπορούμε την έξοδο της σιγμοειδής συνάρτησης να τη μετατρέψουμε σε δυαδική έξοδο με την παρακάτω συνάρτηση που ονομάζεται συνάρτηση κατωφλίου (threshold function):

$$\hat{y} = \begin{cases} 1 & \text{if } \varphi(z) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Αφού είδαμε πως το μοντέλο μαντεύει την κλάση μίας εισόδου τώρα θα εστιάσουμε στο πως προσαρμόζονται οι τιμές του διανύσματος w. Σκοπός είναι να βρούμε τον τρόπο που το w θα λαμβάνει τιμές με τις οποίες μεγιστοποιείται η πιθανότητα το τελικό μοντέλο να προβλέπει τις πραγματικές τιμές. Αυτό επιτυγχάνεται υπολογίζοντας τη μεγιστοποίηση της πιθανοφάνειας:

$$L(w) = P(y|x;w) = \prod_{i=1}^{n} P(y^{(i)}|x^{(i)};w) = \prod_{i=1}^{n} (\varphi(z^{(i)}))^{y^{(i)}} (1 - \varphi(z^{(i)}))^{1-y^{(i)}}$$

Επειδή η πιθανοφάνεια περιέχει εκθετική συνάρτηση είναι πιο εύκολο να υπολογίσουμε τον λογάριθμο της εξίσωσης αυτής:

$$l(w) = logL(w) = \sum_{i=1}^{n} [y^{(i)} log(\varphi(z^{(i)})) + (1 - y^{(i)}) log(1 - \varphi(z^{(i)}))]$$

Μιας και η λογαριθμική είναι γνησίως αύξουσα συνάρτηση, μεγιστοποιώντας την συνάρτηση είναι το ίδιο σαν να μεγιστοποιούμε το ζητούμενο. Η σχέση $l(w)$ μπορεί να γραφτεί

και ως :

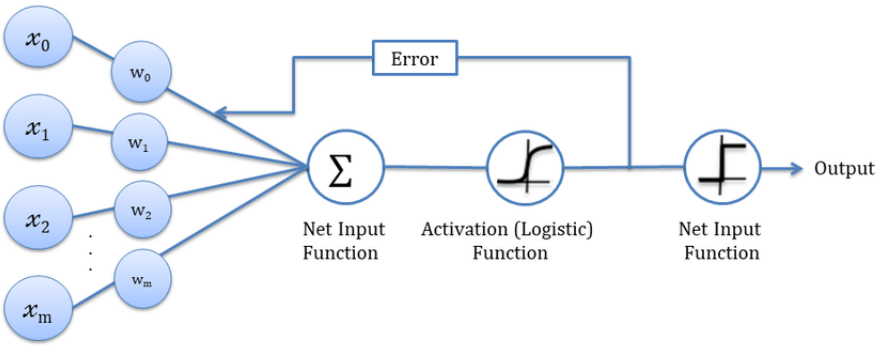
$$J(w) = \sum_{i=1}^{n}[-y^{(i)}log(\varphi(z^{(i)})) - (1 - y^{(i)})log(1 - \varphi(z^{(i)}))]$$

Για την $J(w)$ μπορούμε να την ελαχιστοποιήσουμε (που ισοδυναμεί με την μεγιστοποίηση της $l(w)$ χρησιμοποιώντας τον αλγόριθμο απότομης καθόδου (gradient descent). Άρα καταλήγουμε ότι το w ανανεώνεται σε κάθε γύρο με την σχέση:

$$\Delta w_j = -n\frac{\partial J}{\partial w_j} = n\sum_{i=1}^{n}(y^{(i)} - \varphi(z^{(i)}))x_j^{(i)}$$

$$w_j = w_j + \Delta w_j, \Delta w_j = -n\nabla J(w)$$

Σημείωση: με το σύμβολο $x^{(i)}$ αναφερόμαστε στην i-στη είσοδο για τον αλγόριθμο αντίστοιχα και για $z^{(i)}, y^{(i)}$ [6]. Στο Σχήμα:2.4 φαίνεται σχηματικά ένα παράδειγμα λογιστικής παλινδρόμησης



Σχήμα 2.4: Παράδειγμα Logistic Regression Classifier.

Ridge Classifier

Η μέθοδος του Ridge Classifier, αρχικά ορίζει τις κατηγορίες σε $y = [-1, 1]$, και στην συνεχεία αντιμετωπίζει το πρόβλημα με τη μέθοδο Ridge Regression που είναι η ίδια διαδικασία με την linear regression με την διαφορά ότι η συνάρτηση κόστους είναι διαφορετική και δίνεται από τον παρακάτω τύπο:

$$\text{Ridge:}J(w) = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 + \lambda\sum_{j=1}^{m}w_j^2$$

$$\text{Linear:}J(w) = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Αυξάνοντας την τιμή της παραμέτρου λ, αυξάνουμε την ισχύ της κανονικοποίησης και έτσι συρρικνώνουμε τα βάρη του μοντέλου μας. Λαμβάνοντας υπόψη ότι δεν κανονικοποιούμε τον όρο w_0 [6]. Στην μέθοδο Linear Regression παίρνουμε ως είσοδο ένα διάνυσμα x και ορίζουμε την εξίσωση: $\hat{y} = w \cdot x$. Στόχος μας είναι να μάθουμε τα βάρη της γραμμικής εξίσωσης που θα μας δίνουν την σχέση ανάμεσα στο διάνυσμα x και στην τιμή y. Αυτό

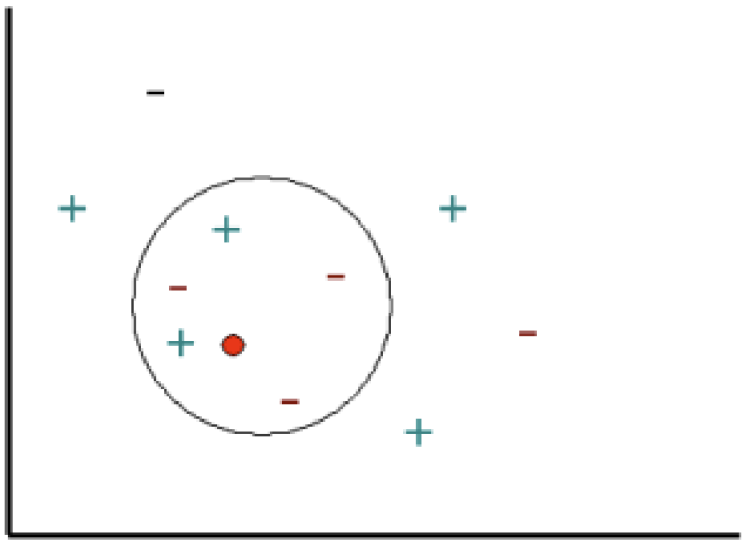
γίνεται ελαχιστοποιώντας την συνάρτηση κόστους. [6]

2.2.2 Μέθοδος K-Κοντινότερων Γειτόνων

Η μέθοδος του κοντινότερου γείτονα είναι μια γενική μέθοδος που μπορεί να εφαρμοστεί στην κατασκευή μοντέλων κατηγοριοποίησης. Η βασική ιδέα είναι πως αν θέλουμε να προβλέψουμε την τιμή μιας καινούριας παρατήρησης x χρησιμοποιώντας το ήδη υπάρχων δείγμα τότε χρησιμοποιούμε για την πρόβλεψη μας την πληροφορία που περιέχουν οι τιμές του δείγματος που μοιάζουν περισσότερο με τη νέα παρατήρηση για την οποία θέλουμε να κάνουμε πρόβλεψη. Η εκτίμηση αυτή δίνεται ως:

$$\hat{y} = \frac{1}{K} \sum_{x \in N_k(x)} y$$

Όπου $N_k(x)$ k είναι το σύνολο που περιέχει τις k πιο κοντινές παρατηρήσεις στο x για το οποίο θέλουμε να κάνουμε πρόβλεψη. Δηλαδή βρίσκουμε ποιες παρατηρήσεις είναι πιο κοντά στην τιμή που θέλουμε να κάνουμε πρόβλεψη παίρνοντας την τιμή με τη μεγαλύτερη συχνότητα στις κοντινότερες παρατηρήσεις. Η παραπάνω ιδέα έχει ως παραμέτρους το k, δηλαδή τον αριθμό των πιο κοντινών με το x παρατηρήσεων, και την απόσταση που υπολογίζουμε για να βρούμε τις κοντινότερες παρατηρήσεις. Υπάρχουν τρόποι να διαλέξουμε το βέλτιστο k, η απόσταση που θα χρησιμοποιήσουμε εξαρτάται από τη φύση του προβλήματος. Σε όλες τις κοντινές παρατηρήσεις ουσιαστικά δίνουμε το ίδιο βάρος. Μια παραλλαγή της μεθόδου είναι να σταθμίσουμε με την απόσταση, δηλαδή πιο όμοιες παρατηρήσεις με μικρή απόσταση να λαμβάνονται περισσότερο υπόψη. Για να γίνει πιο κατανοητός ο αλγόριθμος ας δούμε το παράδειγμα που φαίνεται στο Σχήμα:2.5



Σχήμα 2.5: K-Κοντινότεροι Γείτονες

Αν στο παραπάνω παράδειγμα επιλέξουμε k = 1 τότε είναι προφανές πως σε αυτή την

περίπτωση η μέθοδος του κοντινότερου γείτονα θα κατατάξει το άγνωστο σημείο σαν "συν" (μιας και το κοντινότερο σημείο όπως φαίνεται στο Σχήμα:2.5 ανήκει στην ομάδα των συν). Για k = 2 δεν μπορεί να γίνει κατάταξη γιατί οι δύο πιο κοντινές παρατηρήσεις στο άγνωστο σημείο είναι η μία συν και η άλλη πλην κατά συνέπεια λοιπόν τόσο τα πλην όσο και τα συν έχουν το ίδιο σκορ (ίδιο αριθμό ψήφων). Τέλος αν βάλουμε k = 5 έχουμε ως γειτονική περιοχή, η οποία φαίνεται στο Σχήμα:2.5 με έναν κύκλο. Από τη στιγμή που σε αυτή την περιοχή υπάρχουν 5 σημεία,2 συν και 3 πλην αντίστοιχα, τότε το άγνωστο σημείο χαρακτηρίζεται ως πλην. Η επιλογή του k είναι σημαντική για τον τρόπο λειτουργίας της μεθόδου των k-κοντινότερων γειτονιών. Στην πραγματικότητα, η επιλογή του k θεωρείται ως ένας από τους πιο σημαντικούς παράγοντες του μοντέλου ο οποίος μπορεί να επηρεάσει πολύ την ποιότητα των προβλέψεων. Ένας κατάλληλος τρόπος για να βρούμε τον αριθμό των κοντινότερων γειτονιών k είναι να σκεφτούμε τον αριθμό αυτό σαν μια παράμετρο ομαλότητας (smoothing parameter). Σε κάθε πρόβλημα μικρή τιμή για το k οδηγεί σε πολύ μεγάλη διακύμανση όσον αφορά τις προβλέψεις. Αντίθετα αν δώσουμε στο k μεγάλη τιμή τότε οδηγούμαστε σε ένα μοντέλο με μεγάλη μεροληψία. Από τα παραπάνω προκύπτει πως το k θα πρέπει να είναι αρκετά μεγάλο ώστε να ελαχιστοποιήσει την πιθανότητα λάθους κατάταξης αλλά και αρκετά μικρό ώστε οι k κοντινές παρατηρήσεις να είναι αρκετά κοντά στο άγνωστο σημείο. Έτσι λοιπόν, και όπως με κάθε παράμετρο ομαλότητας (smoothing parameter), υπάρχει μια βέλτιστη τιμή για το k η οποία καταφέρνει να φέρει την ισορροπία μεταξύ μεροληψίας και διακύμανσης στο μοντέλο. Η τιμή του k μπορεί να καθοριστεί με την βοήθεια ενός αλγόριθμου γνωστός ως Cross-Validation [38].

Ο κανόνας των k-Πλησιέστερων Γειτόνων ταξινομεί κάθε μη ετικετοποιημένο δεδομένο βάσει της πλειοψηφίας των k πλησιέστερων γειτόνων του συνόλου εκπαίδευσης. Η απόδοσή του εξαρτάται σε μεγάλο βαθμό από την απόσταση που μετριέται ώστε να εντοπιστούν οι k πλησιέστεροι γείτονες [10]. Ένας σημαντικός παράγοντας για τη λειτουργία του αλγορίθμου των k-Πλησιέστερων Γειτόνων είναι ο καθορισμός της κατάλληλης μετρικής. Οι πλησιέστεροι ταξινομητές χρησιμοποιούν την απλή Ευκλείδεια απόσταση για να μετρήσουν τις ανομοιότητες μεταξύ των διανυσμάτων. Ωστόσο, οι Ευκλείδειες αποστάσεις δεν αξιοποιούν στατιστικές ομαλότητες στα δεδομένα οι οποίες θα μπορούσαν να εκτιμηθούν από ένα μεγάλο σύνολο εκπαίδευσης αποτελούμενο από ετικετοποιημένα παραδείγματα. Η μετρική για την ταξινόμηση ενός αλγορίθμου k-Πλησιέστερων Γειτόνων θα πρέπει να προσαρμόζεται για κάθε πρόβλημα. Γενικά, χρησιμοποιείται η p-μετρική, η οποία έχει την ακόλουθη μορφή:

$$d_p(x,y) = (\sum_{j=1}^{k} |x_j - y_j|^p)^{1/p}$$

από την οποία για p = 1 προκύπτει η μετρική Manhattan, για p = 2 η Ευκλείδια μετρική και για p = ∞ η μετρική Chebysev.

2.2.3 Δένδρα Αποφάσεων

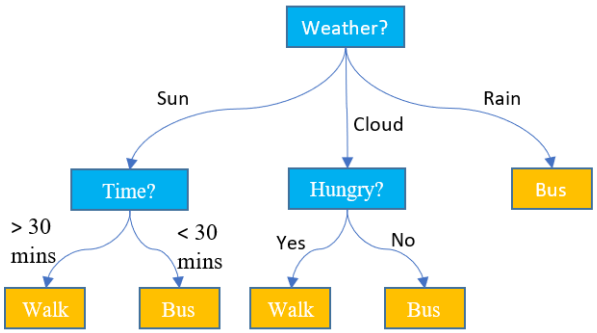
Τα Δένδρα Αποφάσεων είναι μια από τις βασικότερες και πιο δημοφιλείς μεθόδους κατηγοριοποίησης. Βασική λογική της κατασκευής τους είναι η διαδοχική διάσπαση του

συνόλου των παρατηρήσεων σε υποσύνολα. Κριτήριο για τη διάσπαση είναι οι τιμές των μεταβλητών. Η διαδικασία των διαδοχικών διασπάσεων αναπαρίσταται με μια ανεστραμμένη δενδρική δομή. Στην κορυφή βρίσκεται ο κόμβος-ρίζα του δένδρου. Σε κατώτερα επίπεδα βρίσκονται επιπλέον κόμβοι, οι οποίοι συνδέονται με ακμές με άλλα στοιχεία του δένδρου. Στο κατώτερο επίπεδο κάθε κλάδου βρίσκονται τα φύλλα του δένδρου. Ο κόμβος - ρίζα έχει μόνο εξερχόμενες ακμές που τον συνδέουν με στοιχεία του κατώτερου επιπέδου. Οι υπόλοιποι κόμβοι έχουν εισερχόμενες ακμές που τους συνδέουν με τους κόμβους του ανώτερου επιπέδου και εξερχόμενες ακμές που τους συνδέουν με στοιχεία του κατώτερου επιπέδου. Τέλος, τα φύλλα έχουν μόνο εισερχόμενες ακμές, οι οποίες τα συνδέουν με τους κόμβους του ανώτερου επιπέδου. Κάθε κόμβος αντιπροσωπεύει έναν έλεγχο στα δεδομένα και αντίστοιχη διάσπαση τους σε δύο ή περισσότερα υποσύνολα, ανάλογα με το αποτέλεσμα του ελέγχου. Η συνηθέστερη εκδοχή είναι ο έλεγχος να περιλαμβάνει μία μόνο μεταβλητή, έχουν προταθεί ωστόσο αλγόριθμοι όπου σε έναν κόμβο ελέγχονται περισσότερες μεταβλητές. Κάθε ακμή αντιπροσωπεύει ένα αποτέλεσμα του ελέγχου και το αντίστοιχο υποσύνολο των δεδομένων. Τέλος, κάθε φύλλο αντιπροσωπεύει μια απόφαση κατηγοριοποίησης[5]. Ο τρόπος που επιλέγεται ο κόμβος ρίζα, αλλά και οι υπόλοιποι κόμβοι του δέντρου, είναι αρχικά με την εύρεση του κόμβου που διαχωρίζει καλύτερα τα δεδομένα. Ο καλύτερος διαχωρισμός, επιτυγχάνεται με την εύρεση της ανομοιογένειας του κάθε υποψήφιου κόμβου για την εκάστοτε θέση στο δέντρο. Δύο από τις μεθόδους που χρησιμοποιούνται για την εύρεση της ανομοιογένειας, είναι το Gini Index και η εντροπία, με τις ακόλουθες συναρτήσεις αντίστοιχα:

$$entropy(t) = -\sum_{i=0}^{n-1} p(i|t)\log_2 p(i|t)$$

$$Gini(t) = 1 - \sum_{i=0}^{n-1} [p(i|t)]^2$$

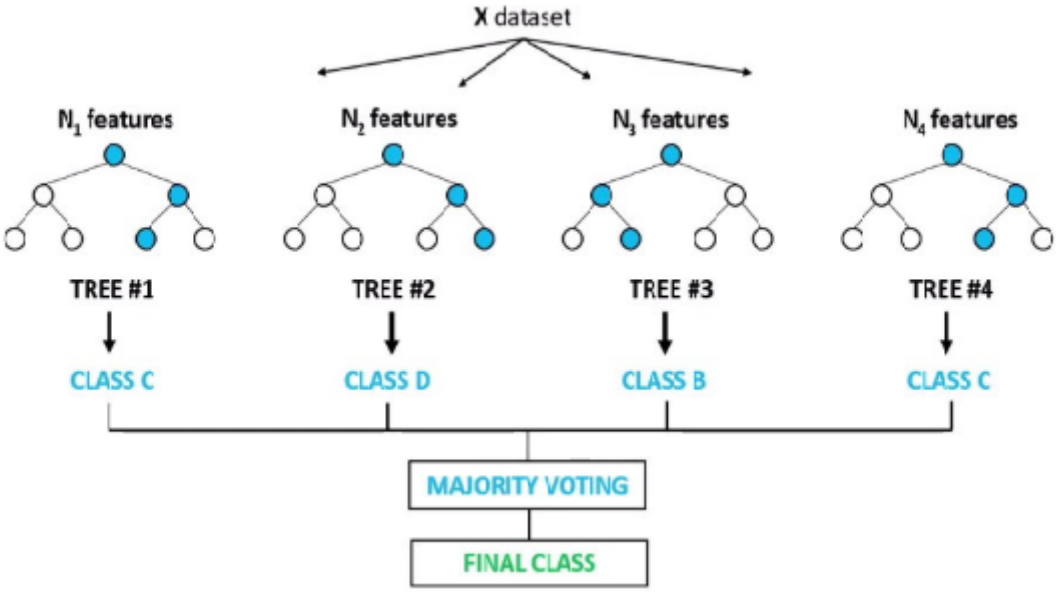
Όπου n το πλήθος των κατηγοριών και $p(i|t)$ το ποσοστό των εγγραφών που ανήκουν στην κατηγορία i σε ένα κόμβο t. Ως αποτέλεσμα, συνήθως επιλέγεται ο κόμβος στον οποίο υπολογίστηκε η χαμηλότερη τιμή εντροπίας ή ο χαμηλότερος δείκτης Gini[5]. Στο Σχήμα:2.6 φαίνεται ένα παράδειγμα δένδρου απόφασης.



Σχήμα 2.6: Δένδρο απόφασης

Random Forest

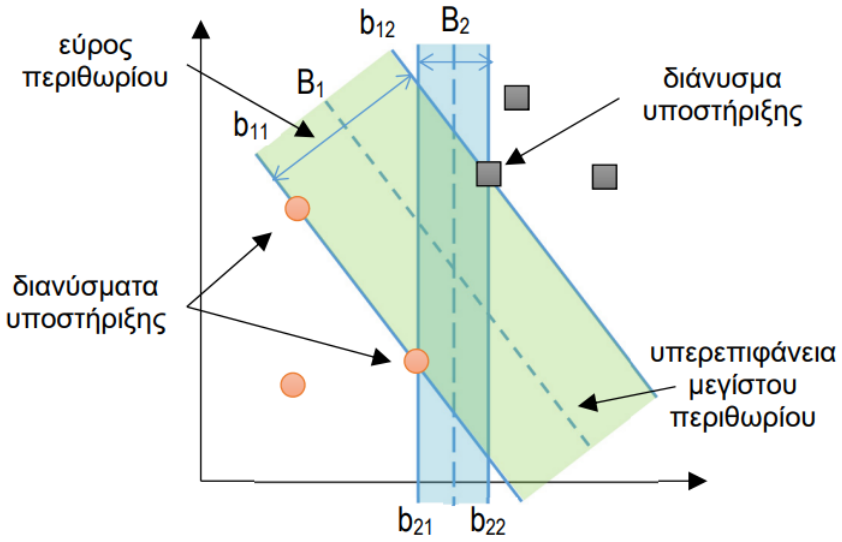
Μια πιο αποδοτική μέθοδος σε σχέση με τα Δένδρα Αποφάσεων, είναι τα τυχαία δάση (Random Forests). Στον αλγόριθμο Random Forest ουσιαστικά εφαρμόζουμε την μέθοδο bagging, σύμφωνα με την οποία χρησιμοποιώντας σε κάθε αντίγραφο τον αλγόριθμο των δένδρων ταξινόμησης παράγουμε πολλά δένδρα απόφασης τα οποία λύνουν το πρόβλημα της μεγάλης διακύμανσης που παρατηρούμε όταν παράγουμε μοναδικό δένδρο απόφασης [39]. Σε δοσμένο σύνολο δεδομένων εκπαίδευσης δημιουργούμε Κ τυχαία δείγματα s_k (για κάθε k=1,2,...,K) των δεδομένων εκπαίδευσης και κατασκευάζουμε δένδρα απόφασης f_k χρησιμοποιώντας κάθε δείγμα s_k σαν σύνολο δεδομένων. Αυτό σημαίνει ότι ξεκινάμε με ένα κενό σύνολο και στην συνέχεια επιλέγουμε τυχαία ένα δείγμα από τα δεδομένα εκπαίδευσης το οποίο το αποθηκεύουμε στο s_k, διατηρώντας το αρχικό σύνολο των δεδομένων εκπαίδευσης. Μετά την εκπαίδευση θα έχουμε Κ δένδρα απόφασης. Η πρόβλεψη για ένα νέο παράδειγμα x΄ λαμβάνεται ως ο μέσος όρος των Κ προβλέψεων[5]. Στο Σχήμα:2.7 φαίνεται ένα παράδειγμα Random Forest.



Σχήμα 2.7: Random Forest

2.2.4 Μηχανές Διανυσμάτων Υποστήριξης (SVM)

Θεωρώντας ένα πρόβλημα δυαδικής ταξινόμησης με θετικά και αρνητικά παραδείγματα, μια επιφάνεια που λειτουργεί ως σύνορο μεταξύ των κλάσεων χαρακτηρίζεται από το γεγονός ότι χωρίζει τα θετικά από τα αρνητικά παραδείγματα. Τέτοια σύνορα υπάρχουν εν γένει πολλά, όπως για παράδειγμα τα B_1 και B_2 στο Σχήμα:2.8. Η μέθοδος SVM επιδιώκει να βρει το σύνορο που απέχει όσο το δυνατόν περισσότερο από τα παραδείγματα των κλάσεων που διαχωρίζει. Η υπέρ-επιφάνεια αυτή ονομάζεται υπέρ-επιφάνεια μεγίστου περιθωρίου και σε γραμμικώς διαχωρίσιμα προβλήματα ορίζεται από έναν πεπερασμένο αριθμό παραδειγμάτων του συνόλου εκπαίδευσης που ονομάζονται διανύσματα υποστήριξης. Με την παραπάνω προσέγγιση, στο πρόβλημα στο Σχήμα:2.8, το σύνορο B_1 είναι το σύνορο με το μέγιστο εύρος περιθωρίου. Επιπλέον, μέσω των συναρτήσεων πυρήνα (kernel functions), οι SVM μπορούν να μετασχηματίσουν τον αρχικό χώρο υποθέσεων έτσι ώστε μη-γραμμικώς διαχωρίσιμα προβλήματα να μετατραπούν σε γραμμικώς διαχωρίσιμα και τελικά να λυθούν με την ίδια μεθοδολογία. Επομένως, το κύριο πλεονέκτημα των SVM είναι ότι όχι μόνο δημιουργούν ένα διαχωριστικό σύνορο μεταξύ των επιμέρους κλάσεων (όπως άλλωστε κάνουν οι περισσότερες μέθοδοι ταξινόμησης), αλλά φροντίζουν αυτό το σύνορο να απέχει όσο το δυνατό περισσότερο από τα παραδείγματα των κλάσεων που διαχωρίζουν.

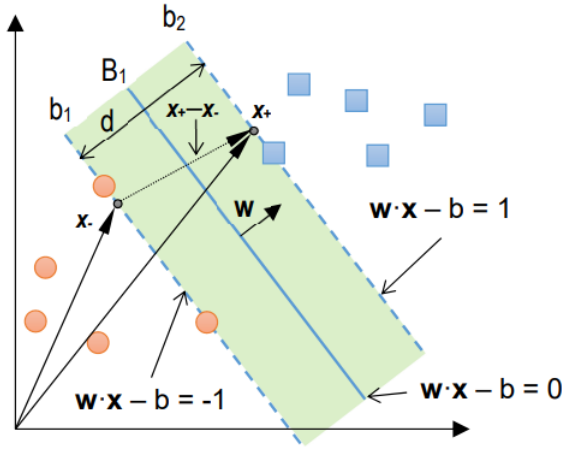


Σχήμα 2.8: Δύο σύνορα απόφασης (διακεκομμένες γραμμές) σε πρόβλημα δυαδικής ταξινόμησης.

Τα σύνορα απόφασης με μεγάλα περιθώρια έχουν κατά βάση μεγαλύτερη ανοχή σε φαινόμενα υπέρ-προσαρμογής. Αν το περιθώριο είναι μικρό, τότε μια μικρή διαταραχή μπορεί να προκαλέσει σημαντική πτώση στην απόδοση του ταξινομητή. Στα μεγάλα περιθώρια, ακόμη και αν τα διανύσματα υποστήριξης είναι παραδείγματα με λίγο θόρυβο, τα σφάλματα γενίκευσης είναι κατά βάση μικρότερα [11].

Γραμμικώς Διαχωρίσιμα Προβλήματα

Έστω ένα σύνολο από πολυδιάστατα σημεία $x(x_1, x_2, ..., x_n)$ για τα οποία γνωρίζουμε ότι ανήκουν στην κλάση (+) (τετράγωνα) ή στην κλάση (–) (κύκλοι), τις οποίες τις κωδικοποιούμε αριθμητικά με 1 και –1 αντίστοιχα.



Σχήμα 2.9: Όρια απόφασης και περιθωρίου σε γραμμική SVM

Θεωρώντας ένα γραμμικώς διαχωρίσιμο πρόβλημα σε δύο διαστάσεις (Σχήμα:2.9), η εξίσωση του συνόρου θα έχει τη μορφή $w_1x_1 + w_2x_2 + b = 0$ ή $wx + b = 0$ αν υιοθετήσουμε διανυσματική γραφή για τα βάρη και το σημείο. Εφαρμόζοντας τη σχέση αυτή

για δύο σημεία α και β που βρίσκονται πάνω στο σύνορο και αφαιρώντας τις δύο σχέσεις προκύπτει:

$$w(x_a - x_b) = 0$$

που σημαίνει ότι το διάνυσμα των βαρών είναι κάθετο στο ζητούμενο σύνορο. Σε γραμμικώς διαχωρίσιμα προβλήματα, μπορούμε να ορίσουμε δύο επιπλέον σύνορα εκατέρωθεν του $w \cdot x + b = 0$ που επίσης χωρίζουν τις δύο κλάσεις και απέχουν όσο γίνεται περισσότερο μεταξύ τους. Η περιοχή που οριοθετούν αυτές οι δύο ευθείες είναι το ζητούμενο μέγιστο περιθώριο (margin) και η ευθεία $w \cdot x + b = 0$ (υπέρ-επιφάνεια) βρίσκεται στη μέση αυτού του περιθωρίου. Επιπλέον, για ένα σημείο z στο χώρο του προβλήματος, η κλάση y του σημείου θα είναι:

$$y = \begin{cases} 1 & \text{if } w \cdot z + b > 0 \\ -1 & \text{if } w \cdot z + b < 0 \end{cases}$$

Έστω x_+ και x_- είναι σημεία στα πάνω και κάτω όρια του περιθωρίου αντίστοιχα (Σχήμα:2.9). Καθώς το διάνυσμα w είναι κάθετο στο ζητούμενο σύνορο, διαιρώντας το με το μήκος του $||w||$ προκύπτει ένα μοναδιαίο διάνυσμα που πολλαπλασιαζόμενο (εσωτερικό γινόμενο) με το διάνυσμα της διαφοράς $x_+ - x_-$, δίνει το ζητούμενο d:

$$(x_+ - x_-) \cdot \frac{w}{||w||} = d$$

Σε αυτή τη σχέση, για το $x_+ \cdot w$, η σχέση υπολογισμού του y δίνει 1-b ενώ για το $x_- \cdot w$ δίνει 1+b. Αντικαθιστώντας στην προηγούμενη σχέση προκύπτει:

$$d = \frac{2}{||w||}$$

Άρα η μεγιστοποίηση του d ισούται με την ελαχιστοποίηση του ||w||. Η διαδικασία εκπαίδευσης της SVM μπορεί να οριστεί μαθηματικά ως η μεγιστοποίηση του d και ισούται με την ελαχιστοποίηση του ||w|| και μπορεί να γραφεί ως:

$$\min_w \frac{||w||}{2}$$

Η τελευταία σχέση μαζί με τους περιορισμούς που ορίζει η έκφραση για τα y, αποτελεί τη μαθηματική διατύπωση της διαδικασίας εκπαίδευσης της SVM. Η επίλυση αυτού του προβλήματος γίνεται με τη βοήθεια των πολλαπλασιαστών Lagrange και προκύπτει ότι τα w και b που ορίζουν το σύνορο απόφασης εξαρτώνται μόνο από διανύσματα (σημεία) που βρίσκονται πάνω στις ευθείες b1 και b2, και από εσωτερικά γινόμενα με αυτά. Τα διανύσματα αυτά ονομάζονται διανύσματα υποστήριξης (support vectors) [11].

Μη Γραμμικώς Διαχωρίσιμα Προβλήματα

Η μέθοδος SVM εφαρμόζεται και όταν τα δεδομένα δεν είναι γραμμικώς διαχωρίσιμα και επομένως δεν υπάρχει κάποιο υπέρ-επίπεδο που να διαχωρίζει πλήρως. Η ιδέα προ-

έρχεται από την παρατήρηση, ότι ναι μεν δεν υπάρχει τέλειο διαχωριστικό υπέρ-επίπεδο στον |T|-διάστατο χώρο, στον οποίο αναπαρίστανται τα κείμενα και οι κατηγορίες με τα αντίστοιχα διανύσματα κειμένων και κατηγοριών, αλλά εντούτοις ενδέχεται να υπάρχει ένα τέλειο διαχωριστικό υπέρ-επίπεδο για τα δεδομένα εκπαίδευσης σε έναν άλλο διανυσματικό χώρο |T|' διαστάσεων, όπου |T|' > |T|. Η μετάβαση στον νέο διανυσματικό χώρο υψηλότερων διαστάσεων, γίνεται με το λεγόμενο "τέχνασμα πυρήνων" (kernel trick). Με αυτό το τέχνασμα παρακάμπτεται ο πλήρης ορισμός των νέων διανυσμάτων στο νέο χώρο, κάτι το οποίο θα ήταν και εξαιρετικά δύσκολο. Άντι για αυτό, χρησιμοποιείται μία συνάρτηση πυρήνων που υπολογίζει την πράξη εσωτερικού διανύσματος μεταξύ των διανυσμάτων στον νέο χώρο. Οι πιο δημοφιλείς πυρήνες είναι οι πολυωνυμικοί [16] $K(x,y) = (xy+1)^P$

Οι Μηχανές Διανυσματικής Υποστήριξης είναι δυαδικοί κατηγοριοποιητές, και έτσι στην περίπτωση προβλημάτων με περισσότερες από δύο κλάσεις, πρέπει να μειωθεί το πρόβλημα σε μια ομάδα πολλαπλών δυαδικών προβλημάτων κατηγοριοποίησης. Ωστόσο, ένα πλεονέκτημα αυτών είναι η ανεκτικότητα που παρουσιάζουν όταν υπάρχει διαφορά στο πλήθος των παραδειγμάτων ανά κλάση αφού οι Μηχανές Διανυσματικής Υποστήριξης δεν επιδιώκουν να μειώσουν το σφάλμα των δεδομένων εκπαίδευσης, αλλά να τα διαχωρίσουν αποτελεσματικά σε ένα χώρο μεγάλης διάστασης.

2.2.5 Αλγόριθμοι Κατηγοριοποίησης Naïve Bayes

Ο απλός ταξινομητής Bayes (Naive bayes Classifier) είναι ένας ταξινομητής βασισμένος στο θεώρημα Bayes με την απλή παραδοχή ότι τα χαρακτηριστικά είναι ανεξάρτητα το ένα από το άλλο. Ακόμη και με αυτή την απλοποίηση, ο ταξινομητής Naïve Bayes έχει αποδείξει την αξία του σε πολλές εφαρμογές, όπως για παράδειγμα αναγνώριση spam. Ένα από τα πλεονεκτήματα του είναι γρήγορος και απλός στην υλοποίηση.

Εστω ένα χαρακτηριστικό διάνυσμα $X = [x_1, x_2, ..., x_n]$ και μια κατηγορική μεταβλητή C_k το θεώρημα Bayes μας λέει ότι:

$$P(C_k|X) = \frac{P(X|C_k)P(C_k)}{P(X)}, για\ \kappa = 1, 2, ..., \mathrm{K}$$

με $P(C_k|X)$ να είναι η εκ των υστέρων πιθανότητα, $P(X|C_k)$ η πιθανοφάνεια, $P(C_k)$ η εκ των προτέρων πιθανότητα της κλάσης ενώ $P(X)$ η εκ των προτέρων πιθανότητα της πρόβλεψης.

Η πιθανοφάνεια $P(X|C_k)$ γράφεται ως εξής κάνοντας χρήση του κανόνα της αλυσίδας:

$$P(X|C_k) = P(x_1, x_2, ..., x_n|C_k) =$$

$$P(x_1|x_2, ..., x_n, C_k)P(x_2|x_3, ..., x_n, C_k)...P(x_n|C_k)$$

τα απλά μοντέλα Bayes υποθέτουν ότι το χαρακτηριστικό x_i
είναι ανεξάρτητο από το χαρακτηριστικό x_j για $i \neq j$ για δοσμένη κλάση.

$$P(x_i|x_{i+1}, ..., x_n|C_k) = P(x_i|C_k)$$

και συνεπώς

$$P(x_1, ..., x_n|C_k) = \prod_{i=1}^{n} P(x_i|C_k)$$

έτσι η πιθανότητα θα γίνει

$$P(C_k|X) = \frac{P(C_k)\prod_{i=1}^{n} P(x_i|C_k)}{P(X)}$$

και αφού το P (X) είναι σταθερό δεδομένου των τιμών εισόδου θα έχουμε

$$P(C_k|X) \propto P(C_k)\prod_{i=1}^{n} P(x_i|C_k)$$

το σύμβολο $\propto$ σημαίνει ότι είναι θετικά ανάλογο προς το δεύτερο μέλος

Έτσι ένα πρόβλημα απλού ταξινομητής Bayes γίνεται ως εξής :

$$\hat{C} = argmax_{C_k} P(C_k)\prod_{i=1}^{n} P(x_i|C_k)$$

όπου $\hat{C}$ είναι η εκτιμώμενη κλάση για το δοσμένο διάνυσμα $\overrightarrow{x}$. Το argmax είναι μια πράξη που βρίσκει το όρισμα που δίνει τη μέγιστη τιμή από μια συνάρτηση στόχο.

Η πιθανοφάνεια $P(x_i|C_k)$ συνήθως διαμορφώνεται χρησιμοποιώντας την ίδια κατηγορία κατανομής πιθανοτήτων. Οι διάφοροι απλοί ταξινομητές Bayes διαφέρουν κυρίως ως προς τις υποθέσεις που κάνουμε σχετικά με την κατανομή του $P(x_i|C_k)$.[12][13]

Multinomial Naïve Bayes

Με ένα πολυωνυμικό μοντέλο, τα δείγματα αντιπροσωπεύουν τις συχνότητες με τις οποίες έχουν δημιουργηθεί από ένα πολυώνυμο $p_1, p_2, ..., p_n$ όπου p_i είναι η πιθανότητα το i συμβάν να πραγματοποιηθεί. Ένα χαρακτηριστικό διάνυσμα $x = (x_1, x_2, ..., x_n)$ τότε είναι ένα ιστόγραμμα, με το x_i να υπολογίζει τον αριθμό των συμβάντων i που παρατηρήθηκαν σε μια συγκεκριμένη περίπτωση. Η πιθανοφάνεια του να παρατηρηθεί ένα ιστόγραμμα x δίνεται από τον τύπο [12][13]:

$$P(x|C_k) = \frac{(\sum_i^n x_i)!}{\prod_i^n x_i!}\prod_i^n p_{p_{ki}^{x_i}}$$

Bernoulli Naïve Bayes

Στο μοντέλο πολλαπλών μεταβλητών Bernoulli, τα χαρακτηριστικά είναι ανεξάρτητες δυαδικές μεταβλητές που περιγράφουν τις τιμές εισόδου. Το μοντέλο αυτό είναι αρκετά δημοφιλές για εργασίες ταξινόμησης εγγράφων. Εάν το x_i είναι δυαδική έκφραση της απουσίας ή εμφάνισης του i όρου από το λεξικό, τότε η πιθανοφάνεια ενός εγγράφου δοσμένης κλάσης C_kθα δίνεται από τον τύπο:

$$p(x|C_k) = \prod_{i}^{n} p_{ki}^{x_i}(1 - p_{ki})^{1-x_i}$$

όπου P_{ki} είναι η πιθανότητα της κλάσης C_k να δημιουργήσει τον όρο x_i. [12][13]

2.2.6 Πολυστρωματικός Νευρώνας

Οι Πολυστρωματικοί Νευρώνες (Multi-Layer Perceptron, MLP) είναι μια από τις κατηγορίες των Τεχνητών Νευρωνικών Δικτύων Εμπρόσθιας Τροφοδοσίας. Αποτελεί ένα σημαντικό εργαλείο μοντελοποίησης, το οποίο εφαρμόζει μια διαδικασία επιβλεπόμενης μάθησης, χρησιμοποιώντας παραδείγματα δεδομένων με γνωστές εξόδους. Ένας Πολυστρωματικός Νευρώνας αποτελείται από τουλάχιστον τρία επίπεδα κόμβων: ένα επίπεδο εισόδου, ένα κρυφό επίπεδο και ένα επίπεδο εξόδου. Εδώ να αναφέρουμε ότι αν έχουμε παραπάνω από ένα κρυφό επίπεδο τότε σχηματίζεται βαθύ τεχνητό νευρωνικό δίκτυο. Τα σήματα διαδίδονται μόνο προς μια κατεύθυνση, δηλαδή από τον κόμβο εισόδου προς τον κόμβο εξόδου. [14] Εκτός από τους κόμβους εισόδου, κάθε άλλος κόμβος είναι ένας νευρώνας που χρησιμοποιεί μια μη γραμμική συνάρτηση ενεργοποίησης. [15] Η βασική διαφορά του Πολυστρωματικού Νευρώνα από έναν Γραμμικό Νευρώνα είναι πως ο πρώτος διαθέτει πολλαπλά επίπεδα και μπορεί να διαχωρίζει παραδείγματα τα οποία δεν είναι γραμμικώς διαχωρίσιμά. [14] Ο όρος Πολυστρωματικός Νευρώνας περιέχει πολλά Perceptrons τα οποία είναι οργανωμένα σε επίπεδα. Κάθε σύνδεση μεταξύ των Perceptrons ενός Πολυστρωματικού Νευρώνα έχει το δικό της βάρος και οι Perceptrons συνήθως χρησιμοποιούν την ακόλουθη συνάρτηση ενεργοποίησης:

$$\varphi(z) = \frac{1}{1 + e^{-z}}$$

Οι Πολυστρωματικοί Νευρώνες είναι χρήσιμοι στην έρευνα, αφού έχουν την ικανότητα να επιλύουν προβλήματα με στοχαστικό τρόπο, γεγονός που επιτρέπει προσεγγιστικές λύσεις για εξαιρετικά πολύπλοκα προβλήματα.

2.3 Natural Language Processing (NLP)

Η επεξεργασία φυσικής γλώσσας είναι ένας κλάδος της επιστήμης της πληροφορικής, της τεχνητής νοημοσύνης και της υπολογιστικής γλωσσολογίας που αφορά στην αλληλεπίδραση μεταξύ υπολογιστών και της ανθρώπινης γλώσσας. Ένας ορισμός που θα μπορούσαμε να δώσουμε ειναι ο ακόλουθος:

Η Επεξεργασία Φυσικής Γλώσσας είναι ένα εύρος από θεωρητικά κίνητρα με υπολογιστικές τεχνικές για την ανάλυση και την αναπαράσταση κειμένων που απαντώνται στη φύση σε ένα ή περισσότερα επίπεδα γλωσσικής ανάλυσης, με σκοπό την επίτευξη επεξεργασίας της γλώσσας παρόμοια με την ανθρώπινη για μια σειρά εργασιών ή εφαρμογών. [17]

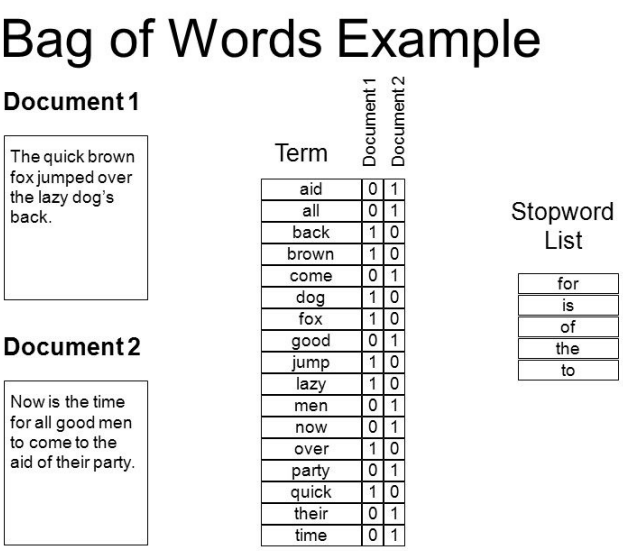
Κύριο αντικείμενο έρευνας αυτής της επιστήμης λοιπόν είναι η κατανόηση της φυσικής γλώσσας με το να παρέχει έτσι τη δυνατότητα στους υπολογιστές να εξάγουν συμπεράσματα από τέτοιου είδους εισόδους δεδομένων, κάτι το οποίο είναι πολύ σημαντικό και

για την παρούσα πτυχιακή εργασία καθώς τη χρησιμοποιήσαμε για να ερμηνευθούν και να κατηγοριοποιηθούν οι ψευδείς ειδήσεις. Η Μηχανική Μάθηση έχει συντελέσει σημαντικά στη διαδικασία της επεξεργασίας φυσικής γλώσσας χρησιμοποιώντας αλγορίθμους μάθησης όλων αυτών των κανόνων που χρειάζονται για να κατηγοριοποιήσουν στοιχεία φυσικής γλώσσας κατόπιν εισόδου σε αυτούς δεδομένων κειμένου. Χρησιμοποιώντας την NLP μέσω των εργαλείων που έχουν αναπτυχθεί, μπορούμε να οργανώσουμε δεδομένα κειμένου, να εκτελέσουμε αυτοματοποιημένες εργασίες και να λύσουμε αρκετά προβλήματα όπως την αυτόματη περίληψη, την ονομαστική αναγνώριση οντοτήτων, την εξαγωγή σχέσεων, την ανάλυση συναισθημάτων με διάφορες διεργασίες όπως το Tokenization[19], δηλαδή τη διαδικασία μετατροπής ενός κειμένου σε tokens, τα οποία είναι λέξεις ή οντότητες που υπάρχουν στο κείμενο. Επειδή το κείμενο είναι η πιο αδόμητη μορφή όλων των διαθέσιμων δεδομένων, τα δεδομένα δεν μπορούν να αναλυθούν χωρίς κατάλληλη προ-επεξεργασία. Η προ-επεξεργασία είναι μία διαδικασία καθαρισμού και τυποποίησης του κειμένου, που το καθιστά χωρίς θόρυβο και έτοιμο για ανάλυση.

Κατά την προ-επεξεργασία ενός κειμένου ακολουθούνται συνήθως τρία στάδια. Το πρώτο στάδιο είναι η *αφαίρεση θορύβου*. Κάθε κομμάτι κειμένου που δεν σχετίζεται με το περιεχόμενο των δεδομένων και την τελική έξοδο μπορεί να θεωρηθεί ως θόρυβος. Για παράδειγμα, λέξεις μιας γλώσσας που χρησιμοποιούνται συχνά και επαναλαμβανόμενα (stopwords), οι διευθύνσεις URL, οι οντότητες των κοινωνικών μέσων π.χ., σημεία στίξης αποτελούν θόρυβο. Συνήθως για την αφαίρεση του θορύβου προετοιμάζουμε ένα λεξικό στο οποίο περιέχονται οι λέξεις που θεωρούμε θόρυβο με στόχο όταν παρουσιάζεται μια λέξη στο κείμενο που υπάρχει, το λεξικό θορύβου την αφαιρεί. Το δεύτερο στάδιο είναι η *Κανονικοποίηση*, το βήμα αυτό μετατρέπει όλες τις διάφορες μορφές μιας λέξης στην ομαλοποιημένη μορφή τους (επίσης γνωστή ως λήμμα). Η Κανονικοποίηση είναι ένα κεντρικό βήμα για τη μείωση των χαρακτηριστικών (ή αλλιώς διαστάσεων) των κειμένων, καθώς μετατρέπει x λέξεις με κοινό λήμμα σε μία, και αυτό το κάνει για όλες τις λέξεις που εμφανίζονται στο κείμενο. Οι πιο συνηθισμένες πρακτικές είναι το Stemming και το Lemmatizer. Το Stemming είναι μια στοιχειώδης διαδικασία βασισμένη στον κανόνα της απογύμνωσης των επιθημάτων από μια λέξη. Αυτό είναι εφικτό με την χρήση γλωσσικών κανόνων και μοτίβων μεταξύ φωνηέντων και σύμφωνων χαρακτήρων. Τα πλεονεκτήματα είναι και πάλι η μείωση των διαστάσεων των δεδομένων και άρα του χρόνου επεξεργασίας. Σε κάποιες περιπτώσεις παρέχει αύξηση της ακρίβειας στην κατηγοριοποίηση κειμένων [18]. Το Lemmatization, από την άλλη πλευρά, είναι μια διαδικασία απόκτησης της ριζικής μορφής της λέξης, που χρησιμοποιεί λεξιλόγιο και μορφολογική ανάλυση. Επίσης μειώνει τις διαστάσεις των δεδομένων και τον χρόνο επεξεργασίας καθώς ακόμα παρέχει αύξηση της ακριβείας στην κατηγοριοποίηση, μόνο που είναι υπολογιστικά πιο απαιτητικό σε σχέση με το stemming κάτι που δεν συμφέρει σε μεγάλα σύνολα δεδομένων [40]. Το τρίτο στάδιο είναι η *τυποποίηση αντικειμένου*. Τα δεδομένα κειμένου περιέχουν συχνά λέξεις ή φράσεις που δεν υπάρχουν σε τυποποιημένα λεξικά. Αυτά τα κομμάτια δεν αναγνωρίζονται από τις μηχανές αναζήτησης και τα μοντέλα. Μερικά από τα παραδείγματα είναι hashtags με συνημμένα λόγια και λέξεις αργκό. Με τη βοήθεια κανονικών εκφράσεων και χειροκίνητων λεξικών δεδομένων, αυτός ο τύπος θορύβου μπορεί να διορθωθεί. [41]

2.3.1 Μετατροπή Κειμένων σε Διανύσματα

Σε ένα σύστημα εξόρυξης γνώσης από κείμενα, αποτελεί απαραίτητο στάδιο η μετατροπή των κειμένων σε διανύσματα. Μια απλή μέθοδος που παρουσιάζει καλά αποτελέσματα σε αρκετές εφαρμογές Μηχανικής Μάθησης είναι το Bag of Words [20]. Σε πρώτο βήμα ο αλγόριθμος εντοπίζει τις διαφορετικές λέξεις που υπάρχουν στο σύνολο των δεδομένων, δηλαδή το λεξιλόγιο. Για το κάθε κείμενο της συλλογής, δημιουργείται ένα διάνυσμα n διαστάσεων όπου n το σύνολο των διαφορετικών λέξεων της συλλογής. Μέσα στο διάνυσμα αυτό περιέχεται ο αριθμός των εμφανίσεων της κάθε λέξης της συλλογής του συγκεκριμένου κειμένου.[20] Ένα παράδειγμα βρίσκεται στο Σχήμα 2.10.



Σχήμα 2.10: Παράδειγμα Bag of Words

Το Tf-Idf [20] είναι ένα μέτρο που χρησιμοποιείται στα πεδία της ανάκτησης πληροφοριών και της Μηχανικής Μάθησης, το οποίο μπορεί να ποσοτικοποιήσει τη σημασία ή τη συνάφεια των αναπαραστάσεων συμβολοσειρών (λέξεις, φράσεις, λήμματα, κ.λπ.) σε ένα έγγραφο ανάμεσα σε μια συλλογή εγγράφων. Το Tf λειτουργεί εξετάζοντας τη συχνότητα ενός συγκεκριμένου όρου που σας απασχολεί σε σχέση με το έγγραφο. Υπάρχουν πολλά μέτρα ή τρόποι καθορισμού της συχνότητας:

1. Πόσες φορές εμφανίζεται η λέξη σε ένα έγγραφο (raw count).

2. Συχνότητα όρου προσαρμοσμένη στο μήκος του εγγράφου (raw count διαιρεμένος με τον αριθμό των λέξεων στο έγγραφο).

3. Λογαριθμικά κλιμακούμενη συχνότητα (π.χ., log (1 + raw count)).

4. Boolean συχνότητα (π.χ., 1 εάν εμφανίζεται ο όρος ή 0 εάν ο όρος δεν εμφανίζεται, στο έγγραφο).

Η αντίστροφη συχνότητα εγγράφου εξετάζει πόσο συνηθισμένη (ή ασυνήθιστη) είναι μια λέξη στη συλλογή. Το IDF υπολογίζεται ως εξής:

$$idf(t, D) = log(\frac{N}{count(d \in D : t \in D)})$$

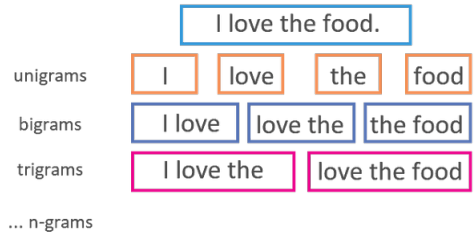
Όπου t είναι ο όρος (λέξη) που ψάχνουμε να μετρήσουμε την συχνότητά του και N είναι ο συνολικός αριθμός των εγγράφων (d) στη συλλογή (D). Ο παρονομαστής είναι ο αριθμός των εγγράφων στα οποία ο όρος t εμφανίζεται. Για να συνοψίσουμε το Tf-Idf συσχετίζει τη σημασία ενός όρου αντιστρόφως με τη συχνότητά του στα έγγραφα. Το Tf μας δίνει πληροφορίες για το πόσο συχνά εμφανίζεται ένας όρος σε ένα έγγραφο και το Idf μας δίνει πληροφορίες για το πόσο συχνά εμφανίζεται ένας όρος στη συλλογή εγγράφων. Πολλαπλασιάζοντας αυτές τις τιμές μαζί μπορούμε να πάρουμε την τελική μας τιμή Tf-Idf. Όσο υψηλότερη είναι η τιμή του μέτρου Tf-Idf τόσο πιο σημαντικός ή σχετικός είναι ο όρος. Καθώς ένας όρος γίνεται λιγότερο σημαντικός ή σχετικός, η βαθμολογία του Tf-Idf θα προσεγγίζει το 0 [20]. Ένα παράδειγμα όρων μιας συλλογής και της σημαντικότητας τους φαίνεται στο Σχήμα 2.11.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

A = "The car is driven on the road"; B = "The truck is driven on the highway" Image from freeCodeCamp - How to process textual data using TF-IDF in Python (https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/)

Σχήμα 2.11: Παράδειγμα Tf-Idf

Τα μοντέλα Bag-of-words/Tf-Idf είναι μια αναπαράσταση εγγράφου χωρίς σειρά, μόνο η εμφάνιση των λέξεων στο κείμενο έχει σημασία. Το μοντέλο N-gram μπορεί να αποθηκεύσει αυτές τις χωρικές πληροφορίες [21]. Η μέθοδος N-gram εντοπίζει τις διαφορετικές n διαδοχικές λέξεις που εμφανίζονται στο σύνολο των δεδομένων, και στη συνέχεια για κάθε κείμενο δημιουργείται όπως και στην μέθοδο Bag of Words ένα διάνυσμα k διαστάσεων, με τη διαφορά ότι το πλήθος των διαστάσεων k είναι οι k διαφορετικές ομάδες n συνεχόμενων λέξεων. Στην Επεξεργασία Φυσικής Γλώσσας τα N-grams χρησιμοποιούνται σε αρκετές εφαρμογές. Μερικά παραδείγματα περιλαμβάνουν την αυτόματη συμπλήρωση προτάσεων, τον αυτόματο ορθογραφικό έλεγχο και σε κάποιο βαθμό, μπορούμε να ελέγξουμε τη γραμματική σε μια δεδομένη πρόταση.[21] Ένα παράδειγμα των N-gram βρίσκεται στο Σχήμα 2.12.



Σχήμα 2.12: Παράδειγμα N-gram

2.4 Μετρικές Αξιολόγησης

Οι μετρικές αξιολόγησης αποτελούν ένα σημαντικό κομμάτι της Μηχανικής Μάθησης καθώς μπορούμε να διακρίνουμε την επιτυχία που έχει κάποιο μοντέλο Μηχανικής Μάθησης και αν αυτό επαρκεί ως προς τις απαιτήσεις που τίθενται. Επιπροσθέτως, μπορούμε να κάνουμε σύγκριση μεταξύ των μοντέλων με την βοήθεια των τιμών που θα ληφθούν στις μετρικές. Παρακάτω παρουσιάζουμε τις μετρικές που χρησιμοποιήθηκαν για το σκοπό της εργασίας.

2.4.1 Confusion Matrix

Ο Confusion Matrix δείχνει τον αριθμό των σωστών και λανθασμένων προβλέψεων που γίνονται για το μοντέλο ταξινόμησης που υλοποιείται σε σχέση με τα πραγματικά αποτελέσματα (τιμή στόχο) στα δεδομένα [23]. Περιέχει επομένως πληροφορίες, σχετικά με την πραγματική και την προβλεπόμενη ταξινόμηση. Οι βέλτιστες λύσεις του μοντέλου έχουν μηδενικές λύσεις περιμετρικά από την κύρια διαγώνιο του πίνακα, ενώ στην κύρια διαγώνιο του πίνακα εμφανίζονται τα ορθά στοιχεία ταξινόμησης, είτε είναι αληθώς θετικά (TruePositive-TP) είτε είναι αληθώς αρνητικά (TrueNegative-TN). Οι περιπτώσεις ψευδώς αρνητικά (FalseNegative-FN) και ψευδώς θετικά (FalsePositive-FP), αντιπροσωπεύουν τις εσφαλμένες ταξινομήσεις για τον υπολογισμό του συνολικού σφάλματος. Γενικότερα, η απόδοση μίας διαδικασίας ταξινόμησης μπορεί να περιγραφεί με ένα confusion matrix, όπως αυτόν που απεικονίζεται στο Σχήμα 2.13 για την περίπτωση ενός προβλήματος ταξινόμησης με δύο κατηγορίες.[23]

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Σχήμα 2.13: Παράδειγμα Confusion Matrix

2.4.2 Accuracy

Accuracy είναι ο λόγος των ορθών προβλέψεων δια του συνόλου των στοιχείων [22]. Σε ένα σύνολο δεδομένων στο οποίο τα TN είναι λίγα, το accuracy score πιθανώς να είναι ψηλό αν ο αλγόριθμος κατηγοριοποιήσει όλα τα δεδομένα ως θετικά. Στην περίπτωση που αποτελεί σημαντικό σφάλμα η λάθος ταξινόμηση των αρνητικών στοιχείων, το accuracy score δεν θα μας δώσει την επιθυμητή ακρίβεια του μοντέλου. Για το λόγο αυτό, υπολογίζονται οι μετρικές Precision και Recall που παρουσιάζονται στις επόμενες παραγράφους. [22]

$$\text{A}ccuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2.4.3 Recall

Recall είναι ο αριθμός των True Positives (TP) διαιρεμένο με το τον αριθμό των True Positives και τον αριθμό των False Negative (FN). Το recall είναι ένα μέτρο της πληρότητας του ταξινομητή. Χαμηλό recall σημαίνει πολλά ψευδώς αρνητικά.[22]

$$Recall = \frac{TP}{TP + FN}$$

2.4.4 Precision

Precision είναι ο αριθμός των True Positives (TP) που διαιρείται από τον αριθμό των True Positives και των False Positive (FP). Το precision είναι ένα μέτρο της ακρίβειας των κατηγοριοποιητών. Χαμηλό precision υποδεικνύει μεγάλο αριθμό ψευδώς False Positive. [22]

$$Precision = \frac{TP}{TP + FP}$$

2.4.5 F1-score

Επειδή τα κριτήρια Precision και Recall δεν αρκούν από μόνα τους για να περιγράψουν την συνολική επίδοση του ταξινομητή συνήθως συνδυάζονται στο κριτήριο F1-score (ή αλλιώς F-measure). Είναι ο αρμονικός μέσος (harmonic average) του precision και του recall.[22]

$$F1score = \frac{2 * \text{Precision} * \text{ Recall}}{\text{Precision + Recall}}$$

Κεφάλαιο 3

Σύνολα δεδομένων

3.1 Πηγή δεδομένων

Η παρούσα πτυχιακή έχει ως στόχο να μελετήσει δεδομένα από το Twitter χρηστών που έχουν να κάνουν με ειδήσεις για την πανδημία του Covid-19. Για το σκοπό αυτό έγινε χρήση δυο συνόλων δεδομένων (dataset), τα οποία εξηγούμε αναλυτικά σε αυτό το κεφάλαιο.

3.1.1 Πρώτο dataset

Το πρώτο dataset που είναι σε μορφή csv, είναι tweets Ελλήνων χρηστών του Twitter από την αρχή της πανδημίας μέχρι το Δεκέμβρη του 2021. Η συλλογή των δεδομένων αυτών από το twitter έγινε με χρήση μιας βιβλιοθήκης στη python που ονομάζεται Snscrape [1] και παρουσίαζεται στην Ενότητα 3.2. Η συγκεκριμένη βιβλιοθήκη δεν χρειάζεται Api keys και επίσης δεν έχει όριο στον αριθμό των tweets που μπορούμε να τραβήξουμε από το Twitter.

Οι δυσκολίες που αντιμετωπίσαμε για να συλλέξουμε αυτά τα δεδομένα ήταν ποικίλες και είχανε αρκετές ιδιομορφίες. Αρχικά, επειδή οι ενεργοί Έλληνες χρήστες του Twitter με συγκεκριμένη θεματολογία (δηλαδή για τον Covid-19) δεν είναι πολλοί σε αριθμό, η συλλογή των δεδομένων ήταν πολύ χρονοβόρα και χρειαζόταν αρκετό ψάξιμο από την πλευρά μας. Στη συνέχεια, μετά από τη συλλογή των δεδομένων, έπρεπε να κατηγοριοποιήσουμε ένα σύνολο από περίπου 5000 tweets χειροκίνητα, σε αληθή και ψευδή, με στόχο να μπορούμε να τα δώσουμε σαν είσοδο στους αλγορίθμους Μηχανικής Μάθησης που περιγράψαμε στο Κεφάλαιο 2. Στο στάδιο της κατηγοριοποίησης που κάναμε, ήταν αρκετά δύσκολο να ξεχωρίσουμε τις αληθινές από τις ψευδείς ειδήσεις καθώς εξαιτίας του θέματος, το οποίο ανήκει στην ιατρική επιστήμη και δεν έχει σχέση με τον κλάδο μας, χρειάστηκε να κάνουμε επιπλέον έρευνα πάνω στο συγκεκριμένο θέμα της πανδημίας. Σίγουρα υπήρχαν αρκετά tweets τα οποία ήταν προφανείς ψευδές ειδήσεις, τουλάχιστον αφουκράζοντας την ιατρική δεοντολογία, αλλά επίσης υπήρχαν και αρκετά διφορούμενα tweets τα οποία δεν ήταν εύκολο να τα κατηγοριοποιήσουμε με αποτέλεσμα πολλά από αυτά να τα αγνοήσουμε. Ένας άλλος λόγος που η συλλογή των δεδομένων ήταν δύσκολη (κυρίως ψάχνοντας για αληθής πληροφορίες πάνω στον covid), είναι ότι ένας μεγάλος

[1] https://github.com/JustAnotherArchivist/snscrape

αριθμός των tweets είχαν και πολιτικό χρώμα με αποτέλεσμα να υπάρχει μια μορφή προπαγάνδας και χρησιμοποίησης της ιατρικής κοινότητας για άλλους σκοπούς. Ο συνολικός αριθμός των tweets που κρατήσαμε από Έλληνες χρήστες για το training των αλγορίθμων είναι 1278 tweets (498 με ετικέτα real και 783 με ετικέτα fake). Τέλος δεν παραθέτουμε κάποιο στιγμιότυπο από το dataset για λόγους προστασίας των προσωπικών δεδομένων.

3.1.2 Δεύτερο dataset

Το δεύτερο dataset που είναι σε μορφή csv προέρχεται από τον γνωστό ιστότοπο Kaggle [2] και αποτελείται από ένα σύνολο 6420 κατηγοριοποιημένων tweets σχετικά με την πανδημία (3360 με ετικέτα real και 3060 με ετικέτα fake). Οι στήλες που περιέχει το συγκεκριμένο dataset είναι το post και μια ετικέτα για true και false. Τα αποτελέσματα που θα εξάγουμε από το dataset αυτό θα τα συγκρίνουμε με αυτά του πρώτου.

3.2 Ανάκτηση και Αποθήκευση δεδομένων

Από κάθε tweet που μας επέστρεφε το Snscrape αποφασίσαμε να κρατήσουμε τα πεδία που φαίνονται στο Σχήμα 3.1 .

Field	*Datatype*	Importance of attributes
followers	*integer*	The number of followers this account currently has.
description	*Text*	The user-defined UTF-8 string describing their account.
verified	*Boolean*	When true, indicates that the user has a verified account.
created	*Text*	The UTC datetime that the user account was created on Twitter.
friends	*Integer*	Number of user's friends.
favorites	*Integer*	The number of Tweets this user has liked in the account's lifetime.
content	*Text*	The actual UTF-8 text of the status update.
likes	*Integer*	Number of likes of the post.
replies	*Integer*	Number of times this Tweet has been replied to.
retweets	*Integer*	Number of times this Tweet has been retweeted.
quotes	*Integer*	Indicates approximately how many times this Tweet has been quoted by users.

Σχήμα 3.1: Πίνακας με την μορφή που αποθηκεύσαμε τα δεδομένα που μας επέστρεφε το Snscrape.

Τα δεδομένα αυτά αποθηκεύτηκαν με τη χρήση ενός script σε ένα csv αρχείο. Παρόλο που κρατήσαμε πεδία που φαίνονται στον πίνακα 3.1, για τον σκοπό της πτυχιακής χρησιμοποιήσαμε για να εκτελέσουμε τους αλγορίθμους Μηχανικής Μάθησης μόνο το πεδίο content που περιέχει το κείμενο ενός tweet. Για την συλλογή των δεδομένων αρχικά έπρεπε να εντοπίσουμε προφίλ χρηστών που ανεβάζουν ειδήσεις (στα ελληνικά) σχετικές

[2] https://www.kaggle.com/datasets/elvinagammed/covid19-fake-news-dataset-nlp?select=Constraint_Train.csv

με τον Covid-19. Αυτό το βήμα επιτεύχθηκε με χειροκίνητη αναζήτηση στο twitter χωρίς χρήση κάποιου script. Έπειτα αφού δημιουργήσαμε μια λίστα με όλα τα προφίλ που καταφέραμε να εντοπίσουμε, χρησιμοποιήσαμε την βιβλιοθήκη Snscrape για να τραβήξουμε όλα τα tweets ενός προφίλ που έχουν δημοσιευτεί μετά την αρχή της πανδημίας. Από τα tweets του κάθε προφίλ που τραβήξαμε, αποθηκεύσαμε τελικά σε json αρχείο μόνο αυτά που σχετίζονται με τον Covid-19. Αυτό το καταφέρνουμε εντοπίζοντας στο κείμενο ενός tweet κάποια σχετική λέξη για την πανδημία. Σχετικές λέξεις για την πανδημία θεωρούνται αυτές που ανήκουν σε ένα λεξικό που έχουμε δημιουργήσει μέσα στο script. Προφανώς το λεξικό έχει μικρό μέγεθος για να είναι γρήγορο το φιλτράρισμα των tweet, παρόλα αυτά η απόδοση του ήταν αρκετά ικανοποιητική και μας εξοικονόμησε αρκετό χρόνο στην συλλογή των δεδομένων. Αφού έχουμε αποθηκεύσει τα δεδομένα στο json αρχείο αφαιρούμε χειροκίνητα tweets που τελικά δεν σχετίζονται με είδηση και ταυτόχρονα τα κατηγοριοποιούμε σε αληθή ή ψευδή. Τέλος γίνεται μια συνένωση των δεδομένων που είναι σε διαφορετικά αρχεία json σε ένα ενιαίο αρχείο csv. Το script που χρησιμοποιήσαμε ακολουθεί παρακάτω:

```
from json import encoder
import snscrape.modules.twitter as twitterScraper
import json
import unicodedata as ud

safe = {
        ”dna”,’mrna’,”εμβολιο”,”εμβλιο0”,”ανεμβλιαστων0”,”pcr”
        ,”covid”,”c0vid”,”εμβολια”,”εμβολιασμο”,”εμβολιασμοι”,”κρουσματα”
        ,”pfizer”,”moderna”,”astrazeneca”,”johnson”,”j&j”,”εμβολιασμενων”
        ,” ιο”,” ιος”,’fda’,”αντισωματα”
        ,”μπολι”,”μπλι0”,”υποχρεωτικοτητα”,”covidpass”
        ,”rapid”,”τσιπακι”,”lockdown”,”λοκνταουν”
        ,”δελτα”,”μεταλλαξη”,’booster’,”dose    ”
        ,”κοβιντ” ,”αντιεμβολιαστες”, ”εμβολιασμοι” ,”υγειονομικοι”
        ,”ανοσια”,”πανδημια”,”ιατρικη”,”sars-cov-2”
        ,”κρουσματα”,”ΕμβολιοΚορωνοιου”,”εμβολιων”,”κορονοιο”
        ,”covid19”,”εμβολιασμενοι”
        ,”εμβλιασμενοι0”,” 5g”,”υποχρεωτικοτητα”
        ,”covid_19”,”πιστοποιητικο”,”πιστοποιητικο”
        ,”μπολι”,”μπλι0”,”υποχρεωτικοτητα”,”covidpass”
        ,”κρονοιου0”,”c0vid-19”,”c0vid_19”,”πανδημια”
        ,”δοση”,”μηυποχρεωτικοτητα-”,”μονοκλωνικα”
        ,”vaccineSideEffects” ,”κορωνοιο”,”deathsfromvaccines”
    }

names = [””]#Profile usernames to scrape.
for name in names:

    scrapper = twitterScraper.TwitterUserScraper(name,False)
    tweets = []
    d = {ord(’\N{COMBINING ACUTE ACCENT}’):None}

    for i,tweet in enumerate(scrapper.get_items()):
        text= ud.normalize(’NFD’,tweet.content).translate(d)
        if(tweet.inReplyToTweetId is  None):
            if(True):
                for word in safe:
```

```
                if(ud.normalize('NFD',word).translate(d) in text):
                    #print(i,word)
                    tweets.append({
                    "followers":tweet.user.followersCount,
                    "description":tweet.user.description,
                    "verified":tweet.user.verified,
                    "created":str(tweet.user.created),
                    "friends":tweet.user.friendsCount,
                    "favorites":tweet.user.favouritesCount,
                    "content": tweet.content,
                    "likes": tweet.likeCount,
                    "replies": tweet.replyCount,
                    "retweets": tweet.retweetCount,
                    "quotes": tweet.quoteCount,
                    })
                    break
    #save the data to csv file
    f = open(name+".json","w",encoding="utf-8")
    j = json.dumps(tweets,ensure_ascii=False)
    f.write(j)
    f.close()
```

3.3 Προετοιμασία Δεδομένων

Κατά την προετοιμασία των δεδομένων εκτελείται προ-επεξεργασία με στόχο να μειωθούν οι διαστάσεις των διανυσμάτων των κειμένων (έχει αναλυθεί στο κεφάλαιο 2.3) και να αποφευχθούν κάποια λάθη, όπως για παράδειγμα η λάθος σύγκριση μεταξύ λέξεων στα κείμενα.

Για την προ-επεξεργασία τα δεδομένα που βρίσκονται στα αρχεία csv τα επεξεργαζόμαστε μέσω μιας βιβλιοθήκης της python την csv [3]. Όλα τα γράμματα του κειμένου μετατρέπονται σε πεζά, στην συνεχεία γίνεται αφαίρεση όλων των link και μη αλφαβητικών χαρακτήρων από το κείμενο, όπως για παράδειγμα τα σημεία στίξης, τόνοι, emoji, tabs και χαρακτήρες αλλαγής γραμμής. Το επόμενο βήμα προ-επεξεργασίας είναι η αφαίρεση των Stop Words μέσω της βιβλιοθήκης nltk [4] που περιέχει όλα τα αγγλικά και ελληνικά Stop Words. Έπειτα εφαρμόζουμε stemming στα δεδομένα, οι stemmer που χρησιμοποιούνται είναι ο SnowballStemmer που βρίσκεται στην βιβλιοθήκη του nltk για τα αγγλικά και ο GreekStemmer που έχει υλοποιηθεί στην βιβλιοθήκη greek-stemmer [5] για τα ελληνικά. Επειδή θέλουμε να δούμε την απόδοση των αλγορίθμων με χρήση stemming και χωρίς, δημιουργούμε δυο ξεχωριστά αρχεία για το κάθε dataset οπού στο ένα αρχείο έχει εφαρμοστεί stemming ενώ στο άλλο δεν έχει εφαρμοστεί. Άρα συνολικά έχουμε 4 αρχεία csv.

Μετά την προ-επεξεργασία, φορτώνουμε τα δεδομένα από τα csv με την βοήθεια της βιβλιοθήκης pandas[6], που μας επιτρέπει τη μετατροπή των αρχείων csv σε DataFrame με τα οποία δουλεύει η βιβλιοθήκη sklearn. Τα δεδομένα του κάθε csv πριν την εφαρμογή της διανυσματοποίησης χωρίστηκαν σε training set και test set σε αναλογία 3:2, με σκοπό

[3] https://docs.python.org/3/library/csv.html
[4] https://www.nltk.org/
[5] https://pypi.org/project/greek-stemmer/
[6] https://pandas.pydata.org/

να εκπαιδευτούν οι αλγόριθμοι με το training set, και να επιβεβαιωθεί η επιτυχία των αλγόριθμων με την χρήση του test set. Ο διαχωρισμός των dataset σε training και test set πριν την διανυσματοποίηση έχει ως σκοπό να μην υπάρχει «διαρροή δεδομένων» μεταξύ του training και test set, δηλαδή το μοντέλο διανυσματοποίησης να μην είναι καθόλου επηρεασμένο από τα δεδομένα επαλήθευσης, τα οποία είναι επιθυμητό να αντιμετωπιστούν τελικά ως νέα, μη γνωστά δεδομένα για μεγαλύτερη ακρίβεια στο μοντέλο. Ο διαχωρισμός των training set και test set, έγινε με την χρήση της βιβλιοθήκης Sklearn.

Το επόμενο βήμα της προετοιμασίας είναι διανυσματοποίηση των δεδομένων, όπου εκτελούνται οι μέθοδοι CountVectorizer [7] και TfidfVectorizer [8] της βιβλιοθήκης Sklearn για μετατροπές των δεδομένων σε μορφές Bag of Words και Tf-Idf αντίστοιχα. Οι μετατροπές CountVectorizer και TfidfVectorizer, εκτελέστηκαν με 3 διαφορετικές τιμές n-gram: 1, 1-2 και 2. Οι διαστάσεις αυξάνονται με την αύξηση του N-gram και μειώνονται με την χρήση Stemming όπως φαίνεται και στον Πίνακα 3.1. Παρόλο που μπορούμε να χρησιμοποιήσουμε τους vectorizer για να κάνουμε την διανυσματοποίηση των κειμένων και να πάρουμε ως έξοδο τα διανύσματα που στην συνέχεια θα τα δώσουμε ως είσοδο στους αλγορίθμους Μηχανικής Μάθησης, προτείνεται σαν καλή πρακτική (για λόγους οργάνωσης και αναγνωσιμότητας του κώδικα) η χρήση της make_pipeline που βρίσκεται στην βιβλιοθήκη sklearn. Η make_pipeline «ενώνει» το μοντέλο του vectorizer και το μοντέλο του αλγορίθμου Μηχανικής Μάθησης και μας δίνει την δυνατότητα να δώσουμε στο νέο μοντέλο που φτιάχνει η make_pipeline τα αρχικά δεδομένα ως είσοδο.

Συνοψίζοντας, για τα δύο σύνολα δεδομένων εκτελέστηκαν 3 μέθοδοι μετατροπών σε διανύσματα, με και χωρίς την εφαρμογή Stemming, με στόχο να γίνει η σύγκριση της επίδρασης των μεθόδων προ-επεξεργασίας των δεδομένων και διανυσματοποίησης όσον αφορά την ακρίβεια του κάθε μοντέλου Μηχανικής Μάθησης. Αφού έχουν εκτελεστεί οι διαδικασίες που περιεγράφηκαν παραπάνω τα δεδομένα είναι έτοιμα για εκτέλεση από τους αλγορίθμους Μηχανικής Μάθησης.

Στον Πίνακα 3.1 φαίνεται η διαφορά στα μεγέθη των λεξικών και για τα δύο datasets που χρησιμοποιήθηκαν. Οι διαφορές προκύπτουν από το αν χρησιμοποιήθηκε ή όχι stemming, καθώς επίσης από τη διαφορετική παραμετροποίηση του N-gram.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer
[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer

Dataset	Stemming	N-gram	Dictionary word count
Dataset 1	No stem	N-gram 1	7918
Dataset 1	Stem	N-gram 1	4718
Dataset 2	No Stem	N-gram 1	12263
Dataset 2	Stem	N-gram 1	8956
Dataset 1	No stem	N-gram 2	20347
Dataset 1	Stem	N-gram 2	18991
Dataset 2	No Stem	N-gram 1-2	59759
Dataset 2	Stem	N-gram 1-2	56080
Dataset 1	No stem	N-gram 1-2	28265
Dataset 1	Stem	N-gram 1-2	23709
Dataset 2	No Stem	N-gram 1-2	72022
Dataset 2	Stem	N-gram 1-2	65036

Πίνακας 3.1: Μεγέθη Λεξικού των dataset

Κεφάλαιο 4

Αποτελέσματα και συμπεράσματα

Σε αυτό το κεφάλαιο παραθέτουμε τα αποτελέσματα της εκτέλεσης των αλγορίθμων Μηχανικής Μάθησης στα dataset με στόχο την ανίχνευση των ψευδών ειδήσεων σχετικά με την πανδημία του Covid-19. Για την σύγκριση τον μοντέλων θα επικεντρωθούμε στο F1-score, γιατί θεωρείται πιο αξιόπιστο σε σχέση με τα μέτρα accuracy, recall και precision. Έχει σημασία να αναφερθεί, ότι στους περισσότερους αλγορίθμους Μηχανικής Μάθησης αλλάξαμε τις default παραμέτρους, καθώς αυτό συνέβαλε σημαντικά στην αύξηση των τιμών των F1-score. Αυτή η αλλαγή έγινε κατόπιν έρευνας που διεξάγαμε σχετικά με το τι επηρεάζει η κάθε παράμετρος αλλά και με αρκετό πειραματισμό παρατηρώντας κάθε φορά τα διαφορετικά αποτελέσματα των εκτελέσεων. Επίσης δίνονται και τα confusion matrix για κάθε εκτέλεση και μέσα από αυτά μπορούν να υπολογιστούν όλες οι μετρικές για την απόδοση των αλγορίθμων. Λόγω χώρου στα confusion matrix δεν διευκρινίζεται το stemming και το N-gram αλλά ακολουθεί την σειρά που έχουμε στους πίνακες αποτελεσμάτων και στα Bar chart.

4.1 Εκτέλεση Αλγορίθμων για Ανίχνευση Ψευδών Ειδήσεων

4.1.1 Percepton Classifier

Στον Πίνακα 4.1 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο perceptron classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Σημειώνεται, ότι έχουμε τεστάρει τα δεδομένα με δυο διαφορετικά μοντέλα. Στην πρώτη περίπτωση (count) κάθε tweet είναι ένα διάνυσμα αποτελούμενο από το πλήθος εμφανίσεων κάθε λήμματος, ενώ στη δεύτερη περίπτωση (Tf-Idf) χρησιμοποιείται το αντίστοιχο μοντέλο για την παραγωγή των διανυσμάτων. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο count καταφέρνει καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο Tf-Idf. Αυτό ταιριάζει με τη διαίσθησή μας καθώς τα tweets δεν αποτελούν δεδομένα με τα συνηθισμένα χαρακτηριστικά εγγράφων, αφού αποτελούνται από μικρό αριθμό λέξεων και σε πολλές περιπτώσεις χρησιμοποιούν τους ίδιους όρους ανάλογα με την μόδα (trend) της ημέρας δημοσίευσης. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.82 F1-score, και ακολουθεί η δεύ-

τερη καλύτερη όταν άλλαξε μόνο σε N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.81 F1-score. Στο Σχήμα 4.1 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.1. Αντίστοιχα, στο Σχήμα 4.3 βλέπουμε τους confusion matrix για το Dataset 1. Από αριστερά προς τα δεξιά και από πάνω προς τα κάτω δείχνουμε το πώς ταξινομήθηκαν στα δεδομένα σε TN/FP/FN/TP ανάλογα με το αν χρησιμοποιήθηκε stemming ή όχι και για τα διαφορετικά N-grams (1 , 2 , ή 1-2), με τη σειρά που φαίνονται στον Πίνακα 3.1. Έτσι, για παράδειγμα ο πάνω αριστερά matrix δείχνει τα αποτελέσματα όταν χρησιμοποιήθηκε count vectorizer με 1 N-gram χωρίς stemming. Αντίστοιχα ο δεύτερος στη σειρά matrix αφορά τα αποτελέσματα όταν χρησιμοποιήθηκε count vectorizer με 1 N-gram με χρήση stemming κοκ.

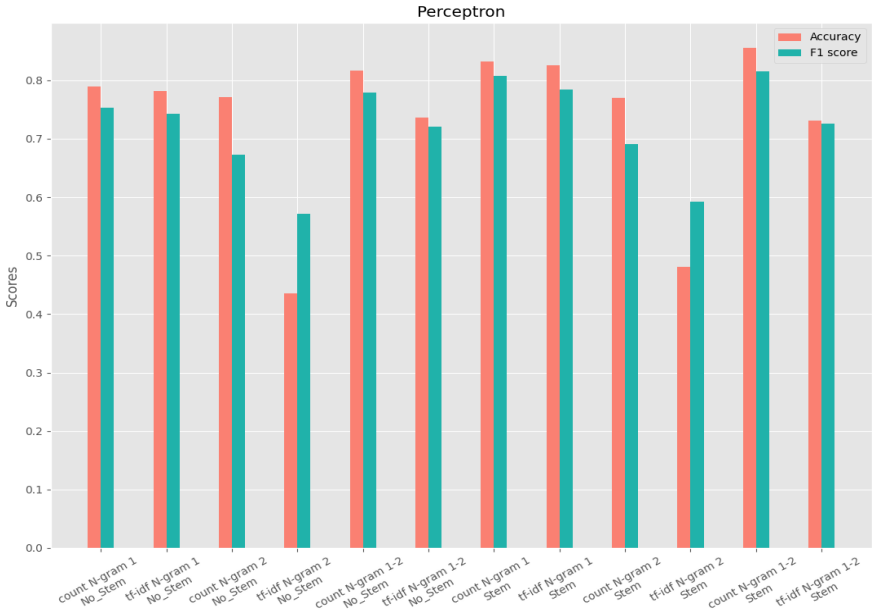
Στον Πίνακα 4.2 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο perceptron classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο count καταφέρνει και καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο Tf-Idf. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, χωρίς stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.89 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1). Στο Σχήμα 4.2 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.2. Αντίστοιχα, στο Σχήμα 4.4 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.7891	0.7534
tf-idf	N-gram 1	No Stem	0.7812	0.7431
count	N-gram 2	No Stem	0.7715	0.6723
tf-idf	N-gram 2	No Stem	0.4355	0.5719
count	N-gram 1-2	No Stem	0.8164	0.7793
tf-idf	N-gram 1-2	No Stem	0.7363	0.7205
count	N-gram 1	Stem	0.832	0.808
tf-idf	N-gram 1	Stem	0.8262	0.7845
count	N-gram 2	Stem	0.7695	0.6911
tf-idf	N-gram 2	Stem	0.4805	0.592
count	N-gram 1-2	Stem	0.8555	0.8159
tf-idf	N-gram 1-2	Stem	0.7305	0.7262

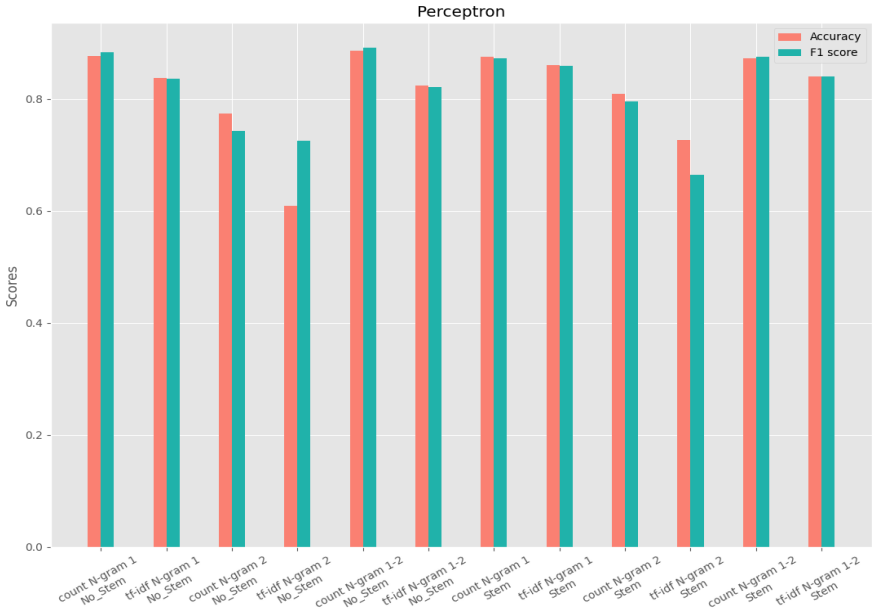
Πίνακας 4.1: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8773	0.8836
tf-idf	N-gram 1	No Stem	0.8372	0.8367
count	N-gram 2	No Stem	0.7734	0.7434
tf-idf	N-gram 2	No Stem	0.609	0.7258
count	N-gram 1-2	No Stem	0.8859	0.8916
tf-idf	N-gram 1-2	No Stem	0.8236	0.8213
count	N-gram 1	Stem	0.875	0.8723
tf-idf	N-gram 1	Stem	0.8602	0.8593
count	N-gram 2	Stem	0.8088	0.7955
tf-idf	N-gram 2	Stem	0.7274	0.6648
count	N-gram 1-2	Stem	0.8727	0.8758
tf-idf	N-gram 1-2	Stem	0.84	0.84

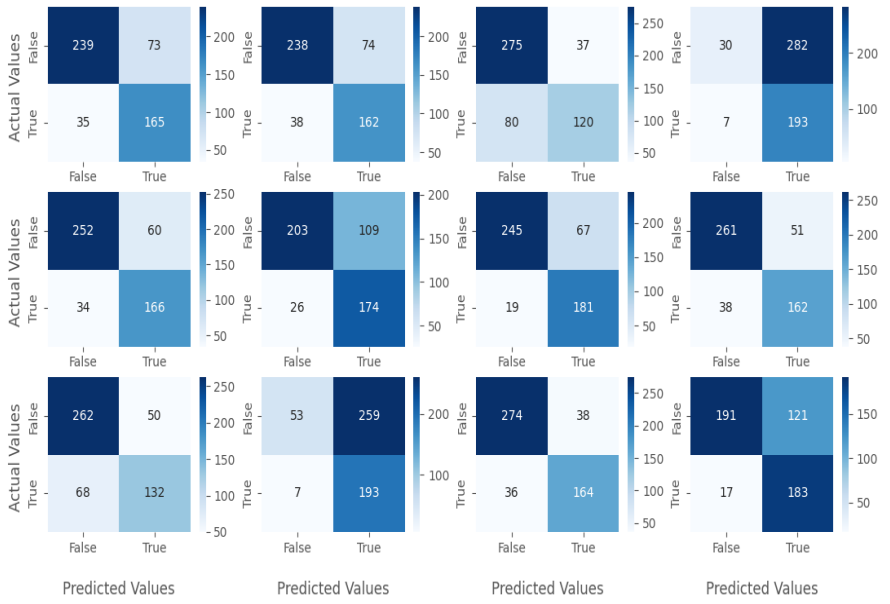
Πίνακας 4.2: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 2



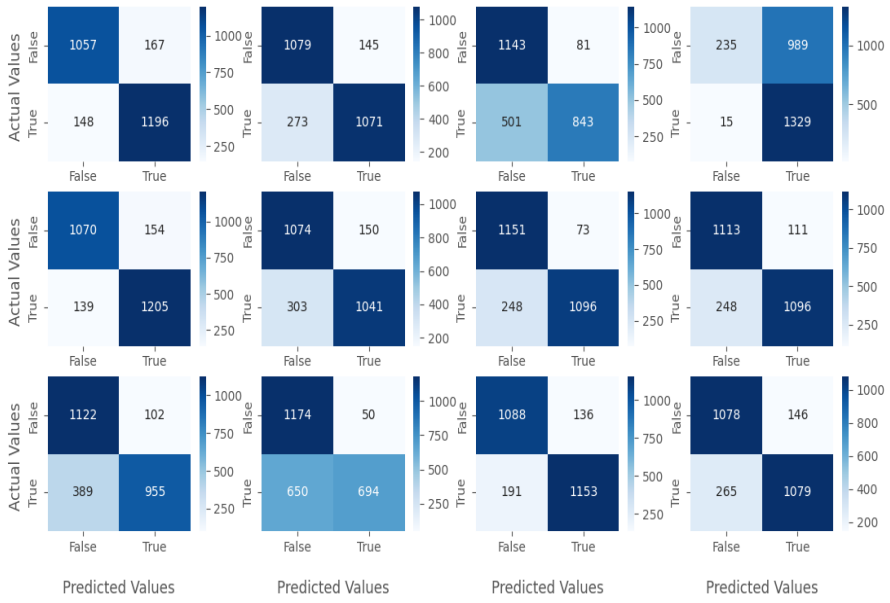
Σχήμα 4.1: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 1



Σχήμα 4.2: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 2



Σχήμα 4.3: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 1



Σχήμα 4.4: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Perceptron Classifier στο Dataset 2

4.1.2 Logistic Regression Classifier

Στον Πίνακα 4.3 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Logistic Regression classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score στις περισσότερες περιπτώσεις σε σύγκριση με το μοντέλο count. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σχεδόν σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score. Στο Σχήμα 4.5 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.3. Αντίστοιχα, στο Σχήμα 4.7 βλέπουμε τους confusion matrix για το Dataset 1.

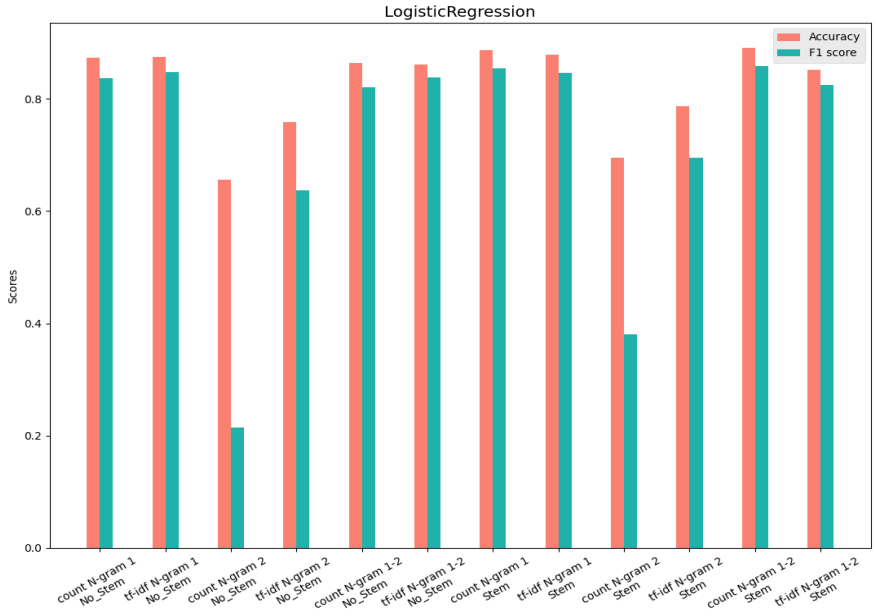
Στον Πίνακα 4.4 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο perceptron classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο count.Επίσης βλέπουμε ότι η χρήση stemming σχεδόν παντού αυξάνει τις επιδόσεις. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.91 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1,2), οπότε και επιτεύχθηκε κοντά στο 0.91 F1-score. Στο Σχήμα 4.6 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.4. Αντίστοιχα, στο Σχήμα 4.8 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.873	0.8363
tf-idf	N-gram 1	No Stem	0.875	0.8476
count	N-gram 2	No Stem	0.6562	0.2143
tf-idf	N-gram 2	No Stem	0.7578	0.6374
count	N-gram 1-2	No Stem	0.8633	0.8205
tf-idf	N-gram 1-2	No Stem	0.8613	0.8383
count	N-gram 1	Stem	0.8867	0.8543
tf-idf	N-gram 1	Stem	0.8789	0.8458
count	N-gram 2	Stem	0.6953	0.381
tf-idf	N-gram 2	Stem	0.7871	0.6947
count	N-gram 1-2	Stem	0.8906	0.8579
tf-idf	N-gram 1-2	Stem	0.8516	0.8241

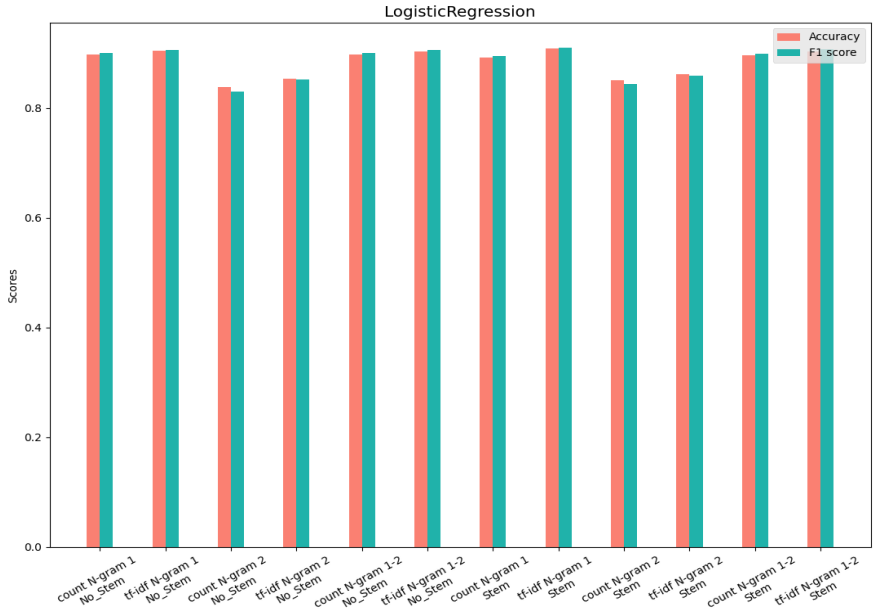
Πίνακας 4.3: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8972	0.8997
tf-idf	N-gram 1	No Stem	0.9042	0.906
count	N-gram 2	No Stem	0.8384	0.8298
tf-idf	N-gram 2	No Stem	0.8536	0.8521
count	N-gram 1-2	No Stem	0.898	0.9008
tf-idf	N-gram 1-2	No Stem	0.903	0.9061
count	N-gram 1	Stem	0.8917	0.8948
tf-idf	N-gram 1	Stem	0.9081	0.9099
count	N-gram 2	Stem	0.8501	0.8433
tf-idf	N-gram 2	Stem	0.861	0.8595
count	N-gram 1-2	Stem	0.8964	0.899
tf-idf	N-gram 1-2	Stem	0.903	0.9065

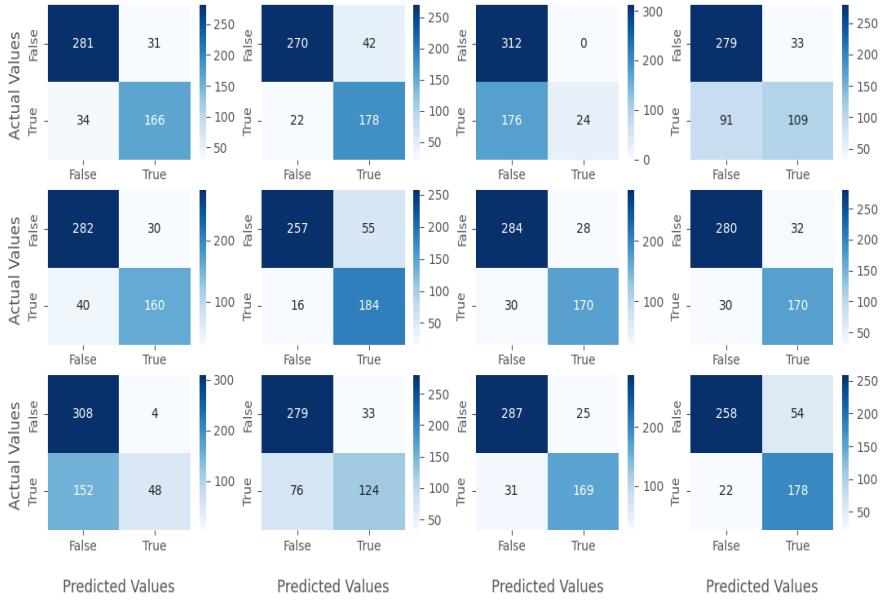
Πίνακας 4.4: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 2



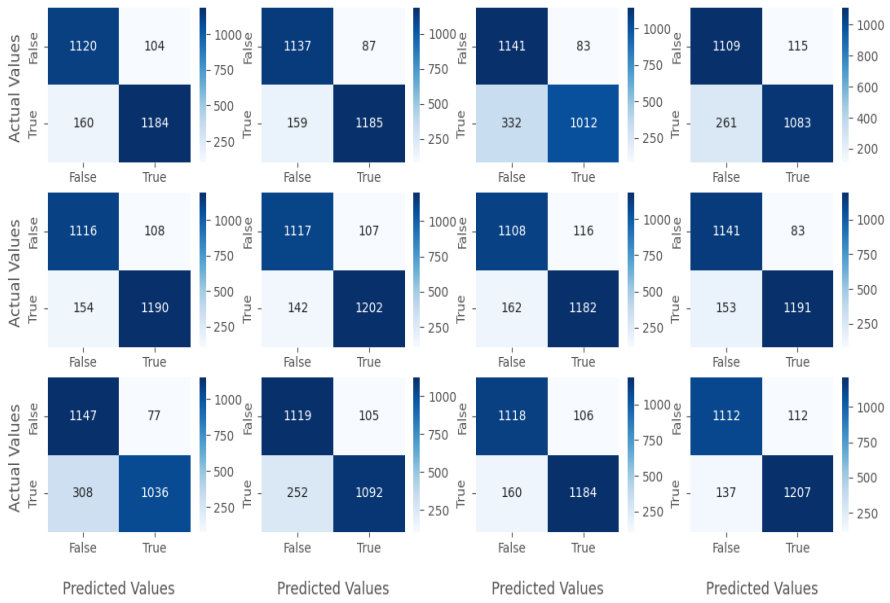
Σχήμα 4.5: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 1



Σχήμα 4.6: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 2



Σχήμα 4.7: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 1



Σχήμα 4.8: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Logistic Regression Classifier στο Dataset 2

4.1.3 Ridge Classifier

Στον Πίνακα 4.5 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Ridge classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score στις περισσότερες περιπτώσεις σε σύγκριση με το μοντέλο count. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1,2), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score. Στο Σχήμα 4.9 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.5. Αντίστοιχα, στο Σχήμα 4.11 βλέπουμε τους confusion matrix για το Dataset 1.

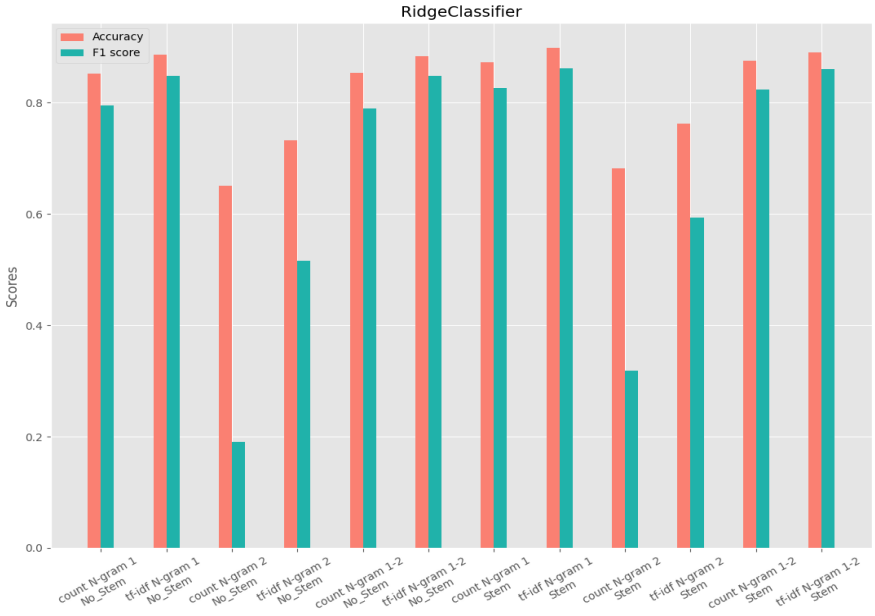
Στον Πίνακα 4.6 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο perceptron classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο count.Επίσης βλέπουμε ότι η χρήση stemming παντού μειώνει τις επιδόσεις .Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, χωρίς stemming και N-gram (1,2), οπότε και επιτεύχθηκε κοντά στο 0.93 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο η χρήση stemming , οπότε και επιτεύχθηκε κοντά στο 0.93 F1-score. Στο Σχήμα 4.10 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.6. Αντίστοιχα, στο Σχήμα 4.12 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8516	0.7946
tf-idf	N-gram 1	No Stem	0.8867	0.8482
count	N-gram 2	No Stem	0.6504	0.19
tf-idf	N-gram 2	No Stem	0.7324	0.5159
count	N-gram 1-2	No Stem	0.8535	0.7899
tf-idf	N-gram 1-2	No Stem	0.8828	0.8485
count	N-gram 1	Stem	0.873	0.8257
tf-idf	N-gram 1	Stem	0.8984	0.8617
count	N-gram 2	Stem	0.6816	0.318
tf-idf	N-gram 2	Stem	0.7617	0.5933
count	N-gram 1-2	Stem	0.875	0.8232
tf-idf	N-gram 1-2	Stem	0.8906	0.8607

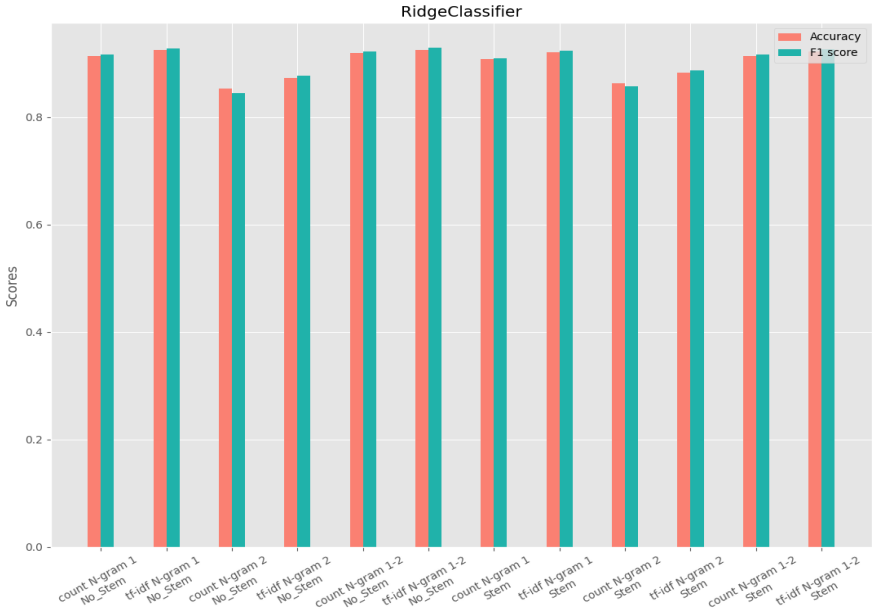
Πίνακας 4.5: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.9139	0.9162
tf-idf	N-gram 1	No Stem	0.9252	0.9279
count	N-gram 2	No Stem	0.8528	0.8447
tf-idf	N-gram 2	No Stem	0.8731	0.8778
count	N-gram 1-2	No Stem	0.919	0.9216
tf-idf	N-gram 1-2	No Stem	0.9252	0.9292
count	N-gram 1	Stem	0.9077	0.9101
tf-idf	N-gram 1	Stem	0.9202	0.9231
count	N-gram 2	Stem	0.8637	0.8574
tf-idf	N-gram 2	Stem	0.8824	0.8866
count	N-gram 1-2	Stem	0.9143	0.9166
tf-idf	N-gram 1-2	Stem	0.9225	0.9267

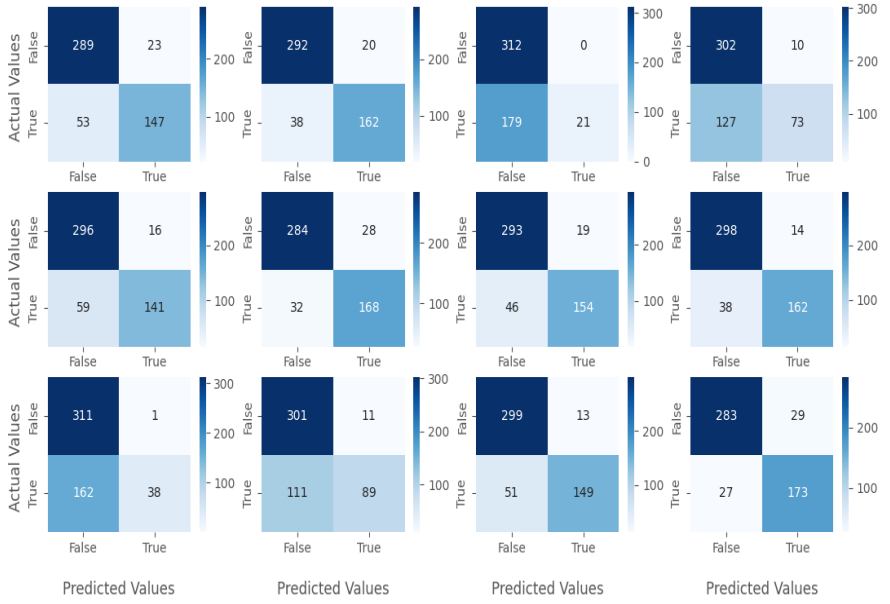
Πίνακας 4.6: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 2



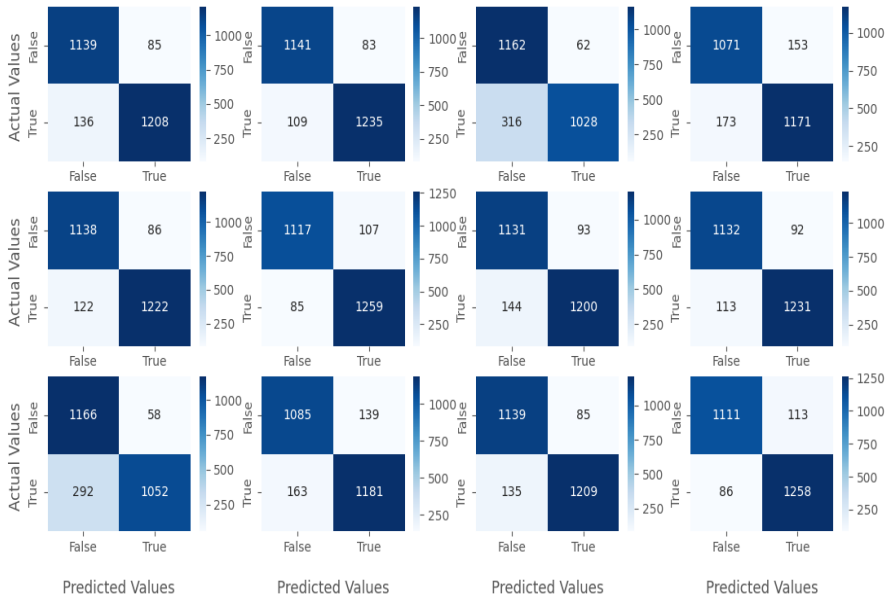
Σχήμα 4.9: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 1



Σχήμα 4.10: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 2



Σχήμα 4.11: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 1



Σχήμα 4.12: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Ridge Classifier στο Dataset 2

4.1.4 k-Nearest Neighbour Classifier

Στον Πίνακα 4.7 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο k-Nearest Neighbour Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρούμε ότι κάποιες τιμές στο F1-Score είναι 0.0 αυτό γίνεται επειδή αν δούμε το confusion matrix στο Σχήμα 4.15 έχουμε τιμή 0 στα TP και αν δούμε τον τύπο του F1-score θα παρατηρήσουμε ότι μηδενίζεται ο παρονομαστής, έτσι βάζουμε συμβολικά το F1-score να γίνεται 0.0. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score σε σύγκριση με το μοντέλο count. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1,2), οπότε και επιτεύχθηκε κοντά στο 0.85 F1-score. Στο Σχήμα 4.14 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.7. Αντίστοιχα, στο Σχήμα 4.15 βλέπουμε τους confusion matrix για το Dataset 1.

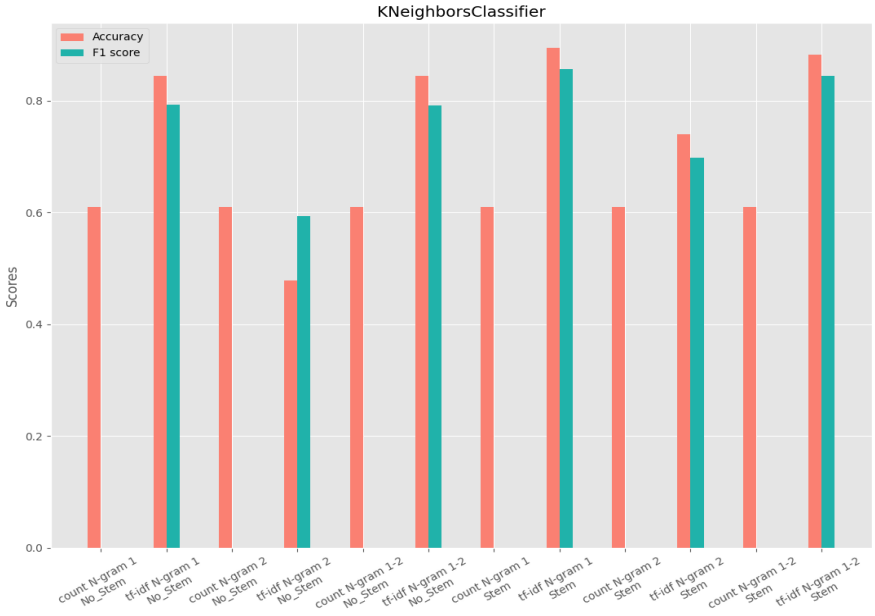
Στον Πίνακα 4.8 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο k-Nearest Neighbour Classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει πολύ καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο count. Επίσης βλέπουμε ότι η χρήση stemming κάποιες φορές δίνει καλύτερα και άλλες χειρότερα αποτελέσματα οπότε δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, χωρίς stemming και N-gram (1,2), οπότε και επιτεύχθηκε κοντά στο 0.90 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο η χρήση stemming, οπότε και επιτεύχθηκε κοντά στο 0.91 F1-score. Στο Σχήμα 4.14 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.8. Αντίστοιχα, στο Σχήμα 4.16 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.6094	0.0
tf-idf	N-gram 1	No Stem	0.8438	0.7927
count	N-gram 2	No Stem	0.6094	0.0
tf-idf	N-gram 2	No Stem	0.4785	0.5936
count	N-gram 1-2	No Stem	0.6094	0.0
tf-idf	N-gram 1-2	No Stem	0.8438	0.7917
count	N-gram 1	Stem	0.6094	0.0
tf-idf	N-gram 1	Stem	0.8945	0.8571
count	N-gram 2	Stem	0.6094	0.0
tf-idf	N-gram 2	Stem	0.7402	0.6984
count	N-gram 1-2	Stem	0.6094	0.0
tf-idf	N-gram 1-2	Stem	0.8828	0.8438

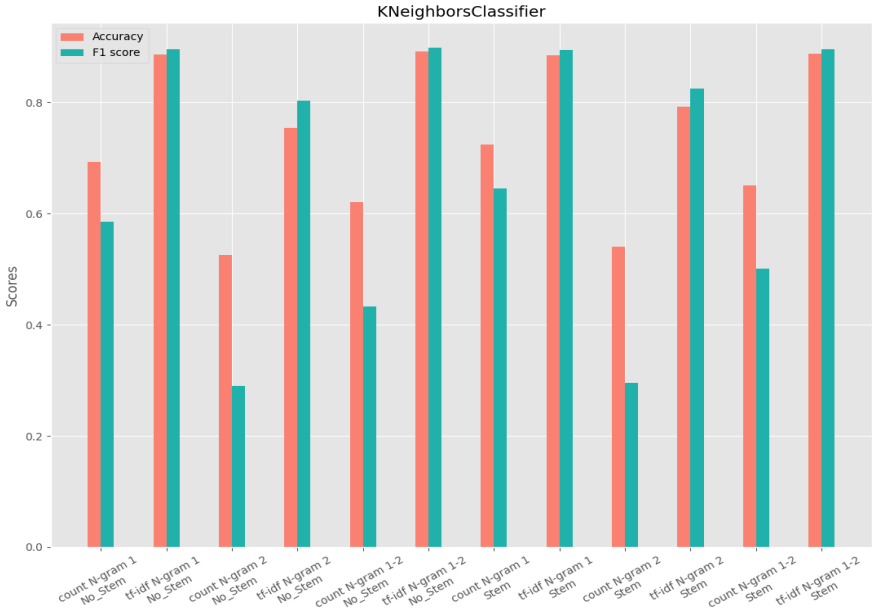
Πίνακας 4.7: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.692	0.5848
tf-idf	N-gram 1	No Stem	0.8863	0.8948
count	N-gram 2	No Stem	0.5249	0.2899
tf-idf	N-gram 2	No Stem	0.7543	0.8028
count	N-gram 1-2	No Stem	0.6207	0.4324
tf-idf	N-gram 1-2	No Stem	0.8906	0.898
count	N-gram 1	Stem	0.7243	0.6453
tf-idf	N-gram 1	Stem	0.8843	0.8934
count	N-gram 2	Stem	0.5397	0.2947
tf-idf	N-gram 2	Stem	0.7917	0.8248
count	N-gram 1-2	Stem	0.6507	0.5003
tf-idf	N-gram 1-2	Stem	0.8875	0.8956

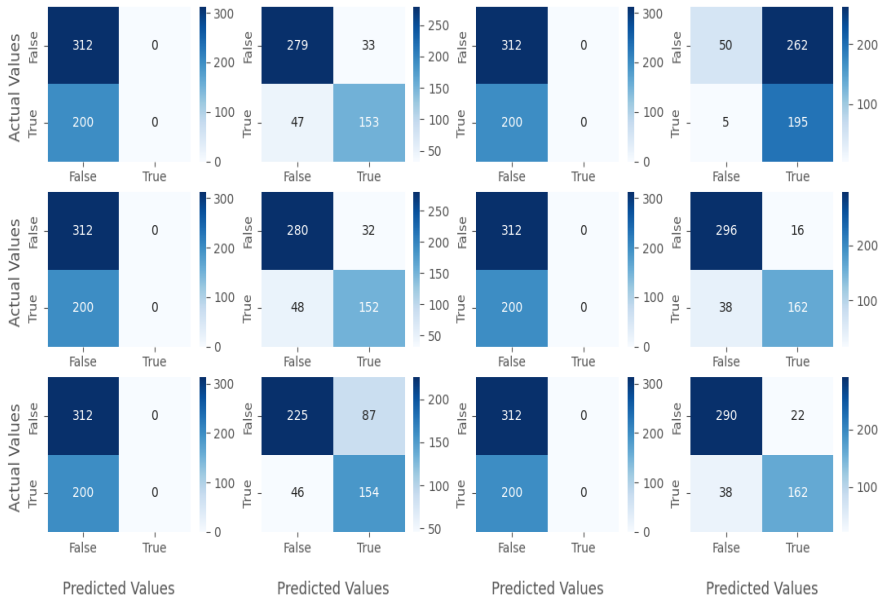
Πίνακας 4.8: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 2



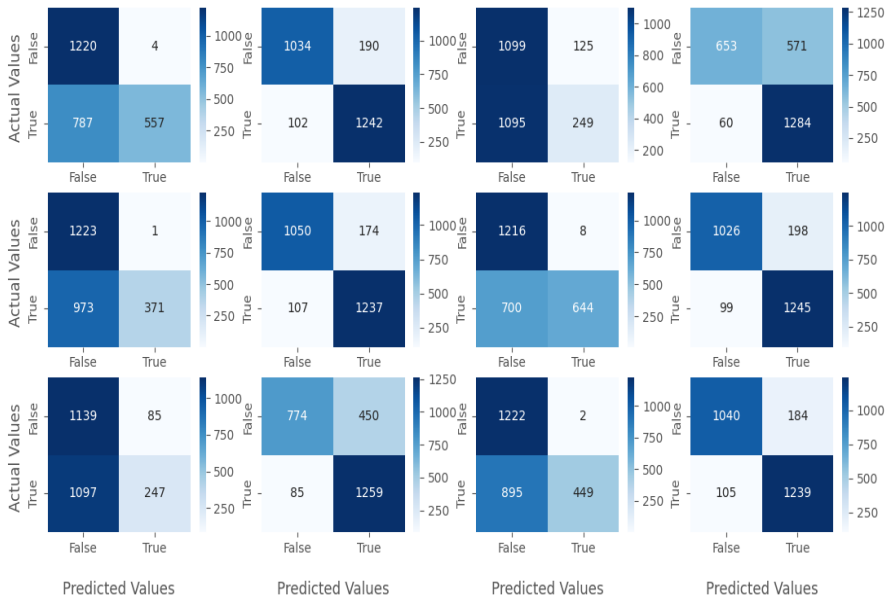
Σχήμα 4.13: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 1



Σχήμα 4.14: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 2



Σχήμα 4.15: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 1



Σχήμα 4.16: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε k-Nearest Neighbour Classifier στο Dataset 2

4.1.5 Decission Tree Classifier

Στον Πίνακα 4.9 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Decission Tree Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα και χειρότερα F1-score σε σχέση με τον μοντέλο count , αρα δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.71 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο το N-gram (1) , οπότε και επιτεύχθηκε κοντά στο 0.71 F1-score. Στο Σχήμα 4.17 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.9. Αντίστοιχα, στο Σχήμα 4.19 βλέπουμε τους confusion matrix για το Dataset 1.

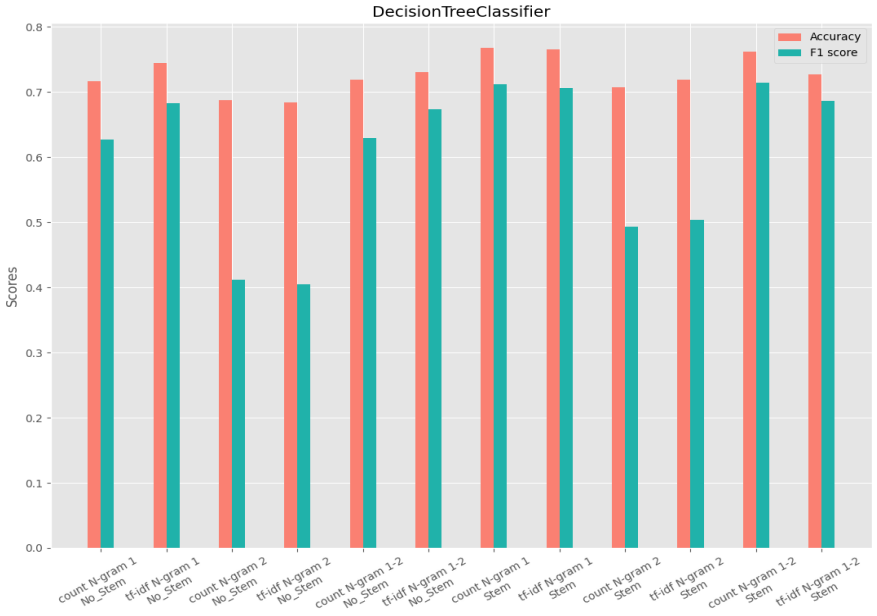
Στον Πίνακα 4.10 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Decission Tree Classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα και χειρότερα F1-score σε σχέση με τον μοντέλο count, άρα δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Επίσης βλέπουμε ότι η χρήση stemming κάποιες φορές δίνει καλύτερα και άλλες χειρότερα αποτελέσματα οπότε δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, χωρίς stemming και N-gram (2), οπότε και επιτεύχθηκε κοντά στο 0.77 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε το μοντέλο Tf-Idf με stemming , οπότε και επιτεύχθηκε κοντά στο 0.76 F1-score. Στο Σχήμα 4.18 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.10. Αντίστοιχα, στο Σχήμα 4.20 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.7168	0.6272
tf-idf	N-gram 1	No Stem	0.7441	0.6828
count	N-gram 2	No Stem	0.6875	0.4118
tf-idf	N-gram 2	No Stem	0.6836	0.4044
count	N-gram 1-2	No Stem	0.7188	0.6289
tf-idf	N-gram 1-2	No Stem	0.7305	0.673
count	N-gram 1	Stem	0.7676	0.7119
tf-idf	N-gram 1	Stem	0.7656	0.7059
count	N-gram 2	Stem	0.707	0.4932
tf-idf	N-gram 2	Stem	0.7188	0.5034
count	N-gram 1-2	Stem	0.7617	0.7136
tf-idf	N-gram 1-2	Stem	0.7266	0.6861

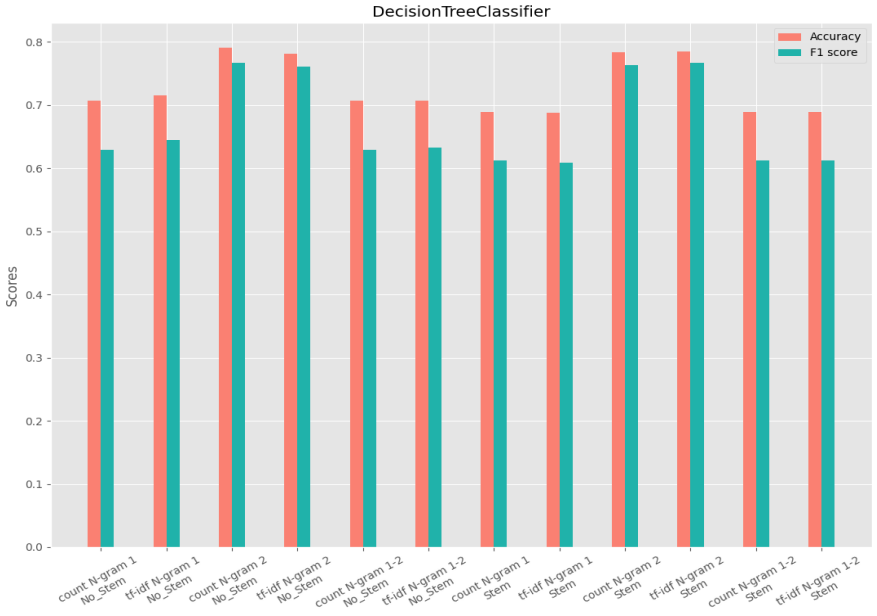
Πίνακας 4.9: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.7076	0.6288
tf-idf	N-gram 1	No Stem	0.715	0.645
count	N-gram 2	No Stem	0.7909	0.767
tf-idf	N-gram 2	No Stem	0.7815	0.7604
count	N-gram 1-2	No Stem	0.7076	0.6288
tf-idf	N-gram 1-2	No Stem	0.7068	0.6332
count	N-gram 1	Stem	0.6893	0.6122
tf-idf	N-gram 1	Stem	0.6881	0.6087
count	N-gram 2	Stem	0.7835	0.7632
tf-idf	N-gram 2	Stem	0.7847	0.7666
count	N-gram 1-2	Stem	0.6893	0.6122
tf-idf	N-gram 1-2	Stem	0.6893	0.6122

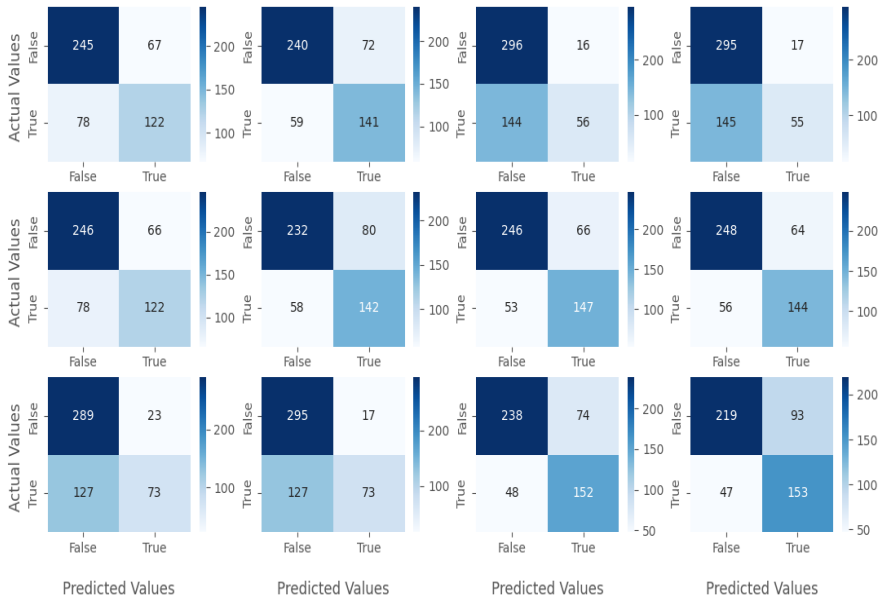
Πίνακας 4.10: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 2



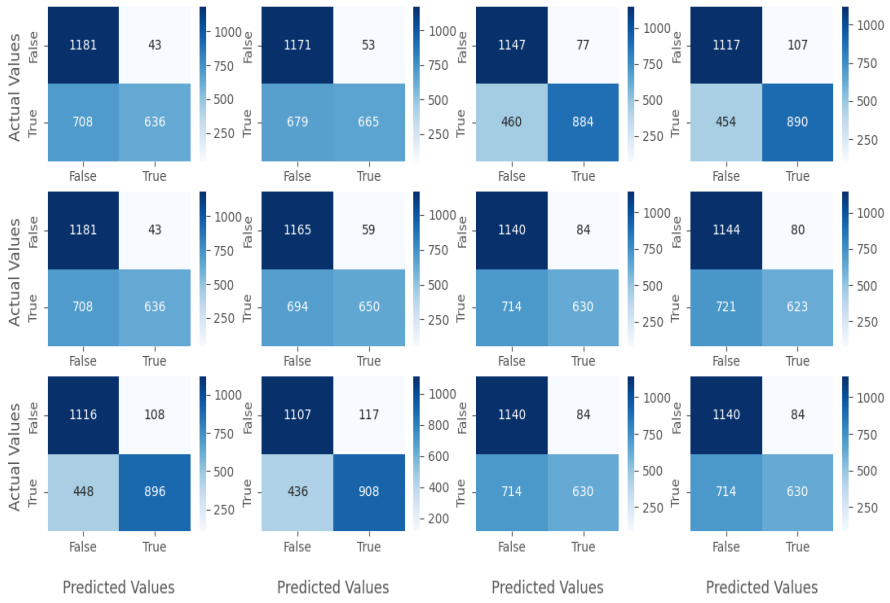
Σχήμα 4.17: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 1



Σχήμα 4.18: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 2



Σχήμα 4.19: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 1



Σχήμα 4.20: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Decission Tree Classifier στο Dataset 2

4.1.6 Random Forest Classifier

Στον Πίνακα 4.11 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Random Forest Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα F1-score σε σχέση με τον μοντέλο count σχεδόν σε όλες τις περιπτώσεις. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.76 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο το μοντέλο count, οπότε και επιτεύχθηκε κοντά στο 0.74 F1-score. Στο Σχήμα 4.21 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.11. Αντίστοιχα, στο Σχήμα 4.23 βλέπουμε τους confusion matrix για το Dataset 1.

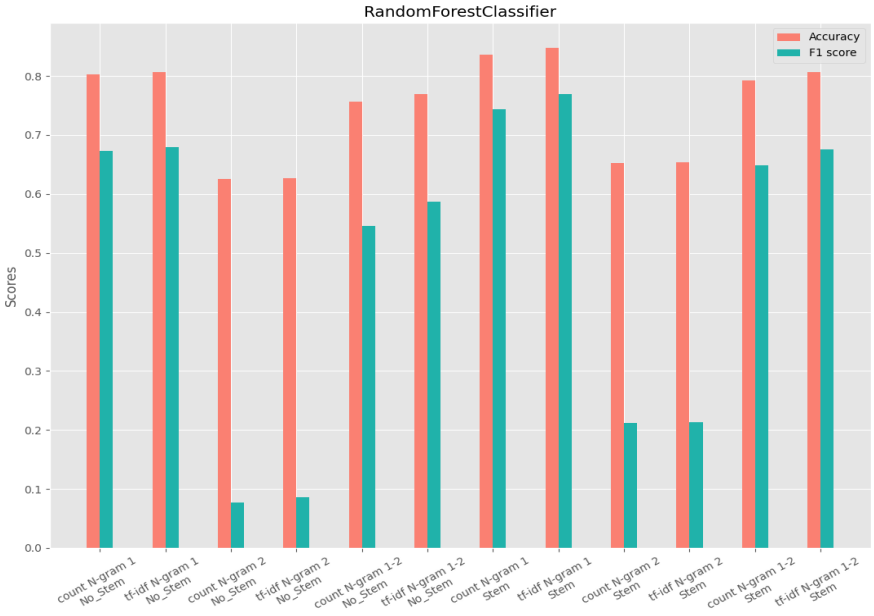
Στον Πίνακα 4.12 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Random Forest Classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα και χειρότερα F1-score σε σχέση με τον μοντέλο count , άρα δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Επίσης βλέπουμε ότι η χρήση stemming κάποιες φορές δίνει καλύτερα και άλλες χειρότερα αποτελέσματα οπότε δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.91 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο το μοντέλο Tf-Idf, οπότε και επιτεύχθηκε κοντά στο 0.91 F1-score. Στο Σχήμα 4.22 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.12. Αντίστοιχα, στο Σχήμα 4.24 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8027	0.6731
tf-idf	N-gram 1	No Stem	0.8066	0.6796
count	N-gram 2	No Stem	0.625	0.0769
tf-idf	N-gram 2	No Stem	0.627	0.0861
count	N-gram 1-2	No Stem	0.7559	0.5455
tf-idf	N-gram 1-2	No Stem	0.7695	0.5874
count	N-gram 1	Stem	0.8359	0.7439
tf-idf	N-gram 1	Stem	0.8477	0.7692
count	N-gram 2	Stem	0.6523	0.2124
tf-idf	N-gram 2	Stem	0.6543	0.2133
count	N-gram 1-2	Stem	0.793	0.649
tf-idf	N-gram 1-2	Stem	0.8066	0.6754

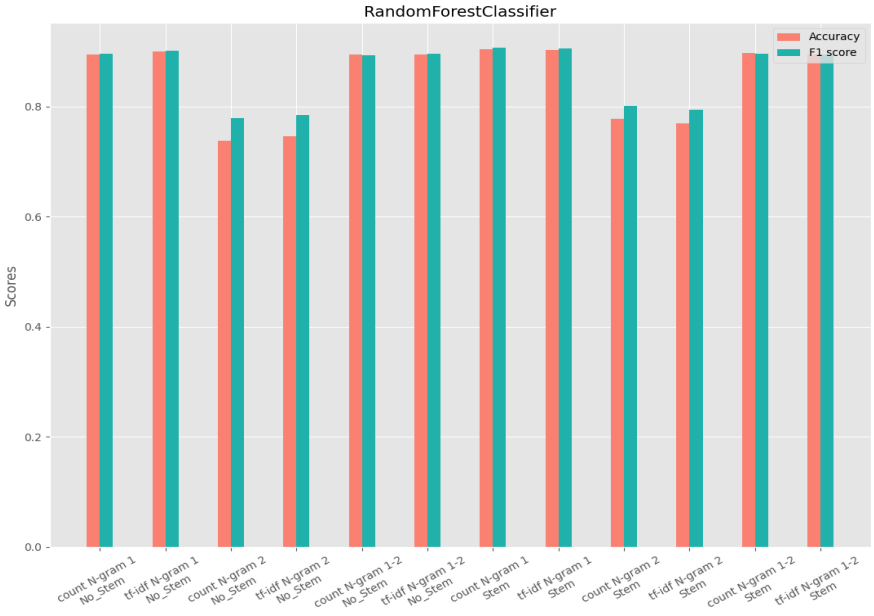
Πίνακας 4.11: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8945	0.8962
tf-idf	N-gram 1	No Stem	0.8999	0.9013
count	N-gram 2	No Stem	0.7375	0.7795
tf-idf	N-gram 2	No Stem	0.7453	0.7846
count	N-gram 1-2	No Stem	0.8937	0.8928
tf-idf	N-gram 1-2	No Stem	0.8949	0.8956
count	N-gram 1	Stem	0.9038	0.9066
tf-idf	N-gram 1	Stem	0.9026	0.9053
count	N-gram 2	Stem	0.7773	0.8007
tf-idf	N-gram 2	Stem	0.7695	0.7939
count	N-gram 1-2	Stem	0.8964	0.8961
tf-idf	N-gram 1-2	Stem	0.8945	0.8948

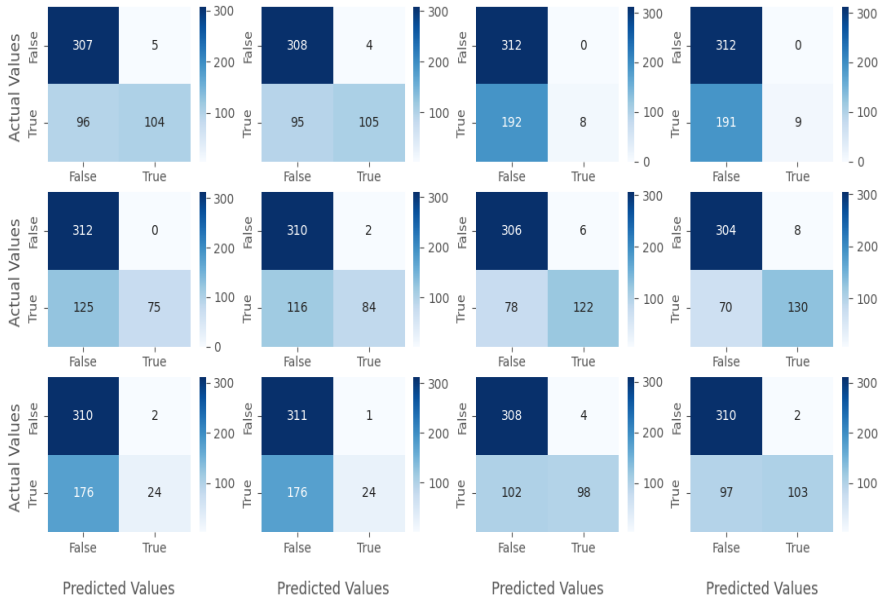
Πίνακας 4.12: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 2



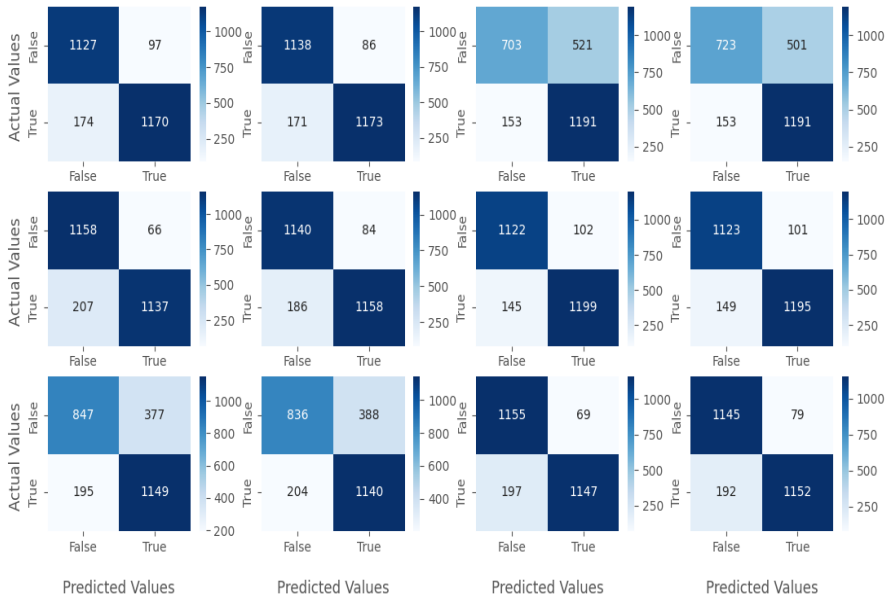
Σχήμα 4.21: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 1



Σχήμα 4.22: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 2



Σχήμα 4.23: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 1



Σχήμα 4.24: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Random Forest Classifier στο Dataset 2

4.1.7 Support Vector Machine Classifier

Στον Πίνακα 4.13 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Support Vector Machine Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score στις περισσότερες περιπτώσεις σε σύγκριση με το μοντέλο count. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σχεδόν σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.87 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο σε N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score. Στο Σχήμα 4.25 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.13. Αντίστοιχα, στο Σχήμα 4.27 βλέπουμε τους confusion matrix για το Dataset 1.

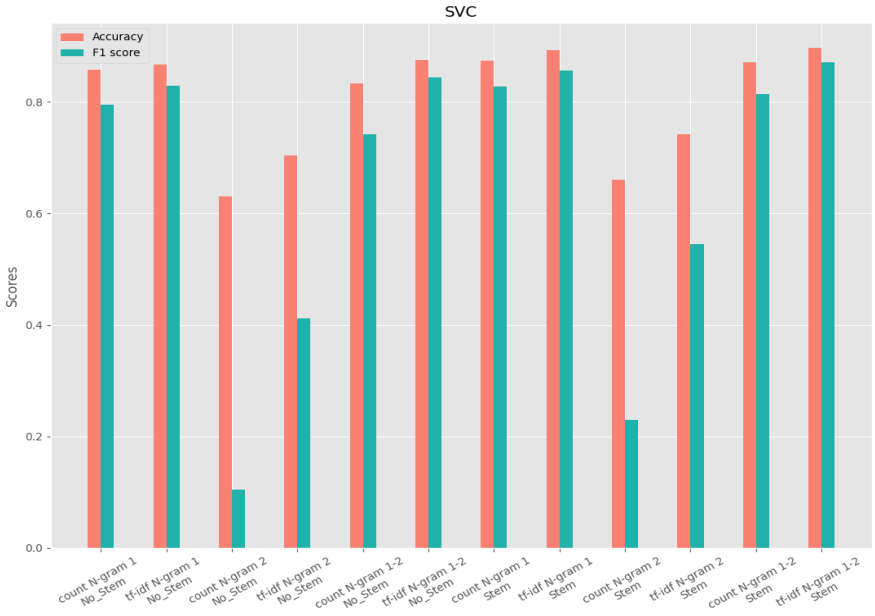
Στον Πίνακα 4.14 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Support Vector Machine Classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο count.Επίσης βλέπουμε ότι η χρήση stemming αυξάνει το F1-score σε κάποιες περιπτώσεις ενώ σε άλλες το μειώνει, οπότε δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, χωρίς stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.93 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο η χρήση stemming, οπότε και επιτεύχθηκε κοντά στο 0.92 F1-score. Στο Σχήμα 4.26 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.14. Αντίστοιχα, στο Σχήμα 4.28 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8574	0.7944
tf-idf	N-gram 1	No Stem	0.8672	0.8283
count	N-gram 2	No Stem	0.6309	0.1043
tf-idf	N-gram 2	No Stem	0.7031	0.4109
count	N-gram 1-2	No Stem	0.832	0.741
tf-idf	N-gram 1-2	No Stem	0.875	0.8439
count	N-gram 1	Stem	0.873	0.8267
tf-idf	N-gram 1	Stem	0.8926	0.8564
count	N-gram 2	Stem	0.6602	0.2301
tf-idf	N-gram 2	Stem	0.7422	0.5448
count	N-gram 1-2	Stem	0.8711	0.8136
tf-idf	N-gram 1-2	Stem	0.8965	0.8704

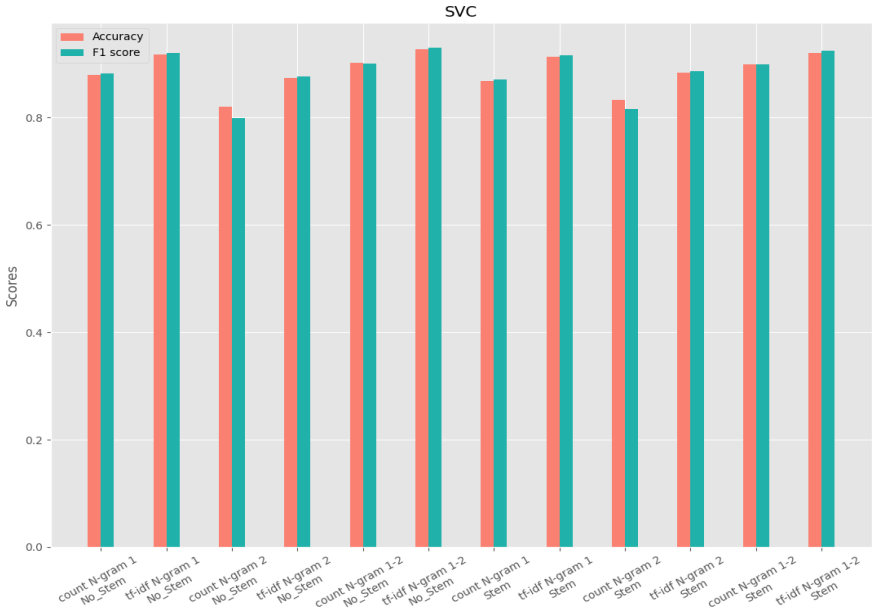
Πίνακας 4.13: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8797	0.8816
tf-idf	N-gram 1	No Stem	0.9174	0.9202
count	N-gram 2	No Stem	0.8197	0.799
tf-idf	N-gram 2	No Stem	0.8734	0.8767
count	N-gram 1-2	No Stem	0.9019	0.9009
tf-idf	N-gram 1-2	No Stem	0.9264	0.9299
count	N-gram 1	Stem	0.868	0.8706
tf-idf	N-gram 1	Stem	0.9136	0.9162
count	N-gram 2	Stem	0.8333	0.816
tf-idf	N-gram 2	Stem	0.8828	0.8858
count	N-gram 1-2	Stem	0.8995	0.899
tf-idf	N-gram 1-2	Stem	0.9198	0.9237

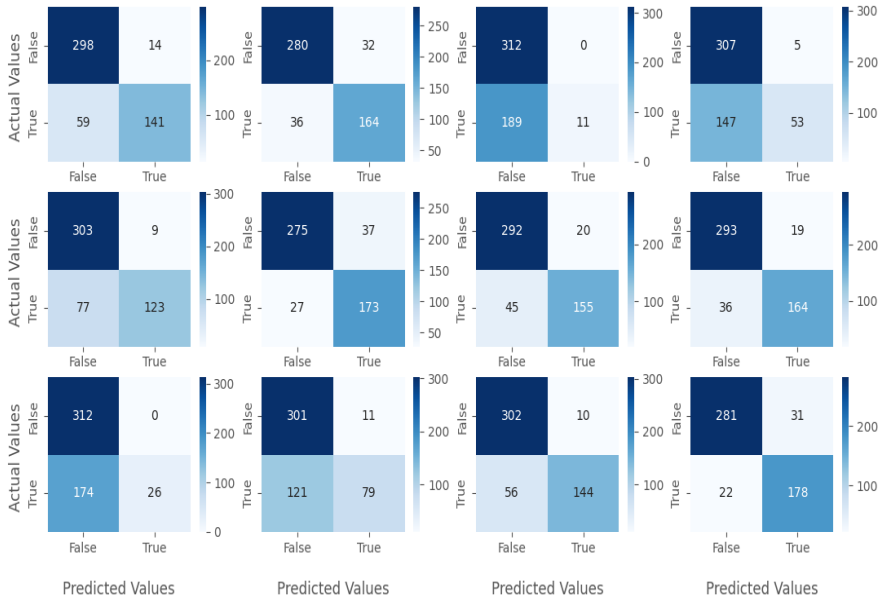
Πίνακας 4.14: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 2



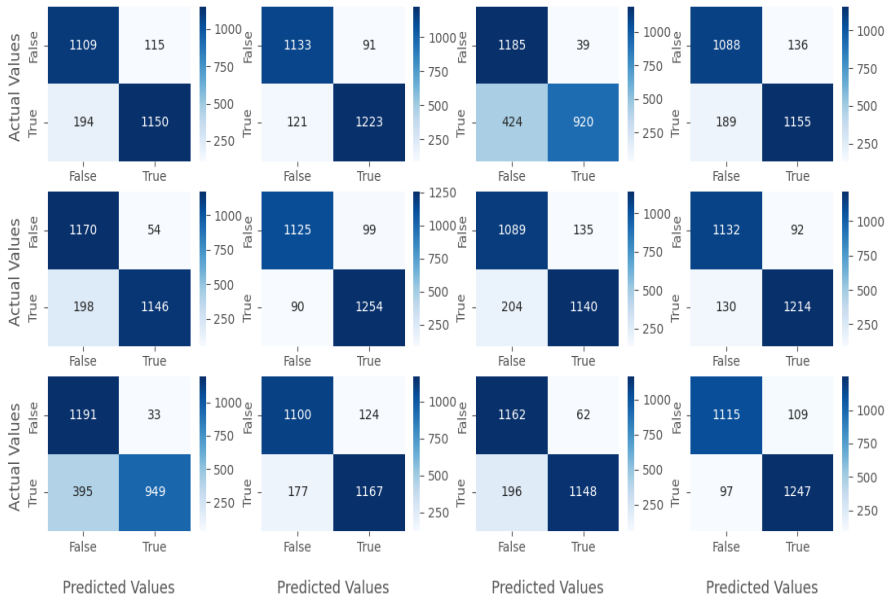
Σχήμα 4.25: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 1



Σχήμα 4.26: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 2



Σχήμα 4.27: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 1



Σχήμα 4.28: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Support Vector Machine Classifier στο Dataset 2

4.1.8 Multinomial Naïve Bayes Classifier

Στον Πίνακα 4.15 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Multinomial Naïve Bayes Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο count καταφέρνει καλύτερο F1-score σχεδόν σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο Tf-Idf. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε στο μοντέλο count και με χρήση N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.86 F1-score. Στο Σχήμα 4.29 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.15. Αντίστοιχα, στο Σχήμα 4.31 βλέπουμε τους confusion matrix για το Dataset 1.

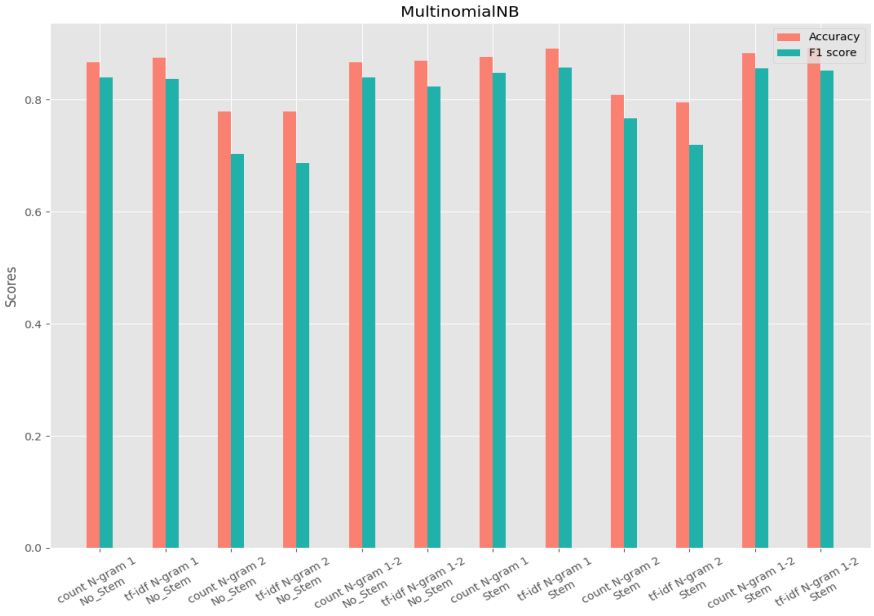
Στον Πίνακα 4.16 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Multinomial Naïve Bayes classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf καταφέρνει καλύτερο F1-score σε όλες τις περιπτώσεις σε σύγκριση με το μοντέλο count.Επίσης βλέπουμε ότι η χρήση stemming δεν σημαντικά αυξάνει τις επιδόσεις καθώς επίσης σε μερικές περιπτώσεις παρατηρούμε και μείωση του F1-score. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, χωρίς stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.93 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο η χρήση stemming, οπότε και επιτεύχθηκε κοντά στο 0.92 F1-score. Στο Σχήμα 4.30 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.16. Αντίστοιχα, στο Σχήμα 4.32 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8672	0.8404
tf-idf	N-gram 1	No Stem	0.875	0.8367
count	N-gram 2	No Stem	0.7793	0.7034
tf-idf	N-gram 2	No Stem	0.7793	0.687
count	N-gram 1-2	No Stem	0.8672	0.8396
tf-idf	N-gram 1-2	No Stem	0.8691	0.8232
count	N-gram 1	Stem	0.877	0.8475
tf-idf	N-gram 1	Stem	0.8906	0.8579
count	N-gram 2	Stem	0.8086	0.7667
tf-idf	N-gram 2	Stem	0.7949	0.72
count	N-gram 1-2	Stem	0.8828	0.8565
tf-idf	N-gram 1-2	Stem	0.8926	0.8525

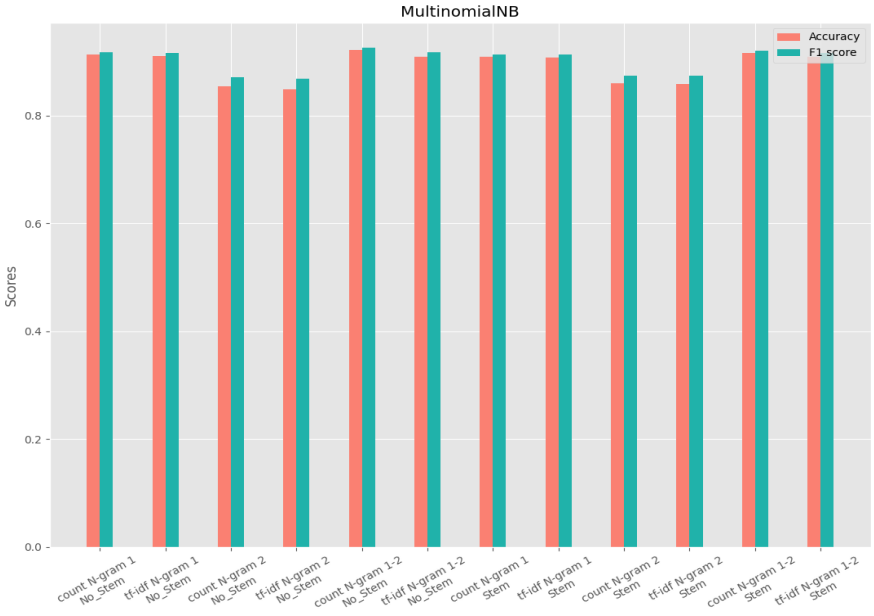
Πίνακας 4.15: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.9132	0.9174
tf-idf	N-gram 1	No Stem	0.9104	0.9165
count	N-gram 2	No Stem	0.8551	0.8719
tf-idf	N-gram 2	No Stem	0.8493	0.8681
count	N-gram 1-2	No Stem	0.9221	0.9263
tf-idf	N-gram 1-2	No Stem	0.91	0.9172
count	N-gram 1	Stem	0.91	0.9138
tf-idf	N-gram 1	Stem	0.9077	0.9135
count	N-gram 2	Stem	0.8606	0.8749
tf-idf	N-gram 2	Stem	0.859	0.8748
count	N-gram 1-2	Stem	0.9171	0.9213
tf-idf	N-gram 1-2	Stem	0.9093	0.9159

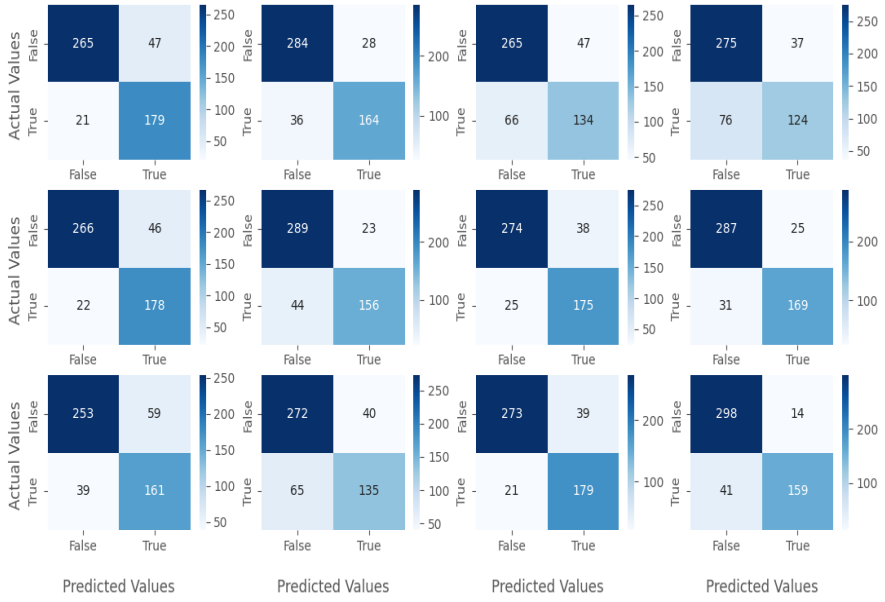
Πίνακας 4.16: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes Classifier στο Dataset 2



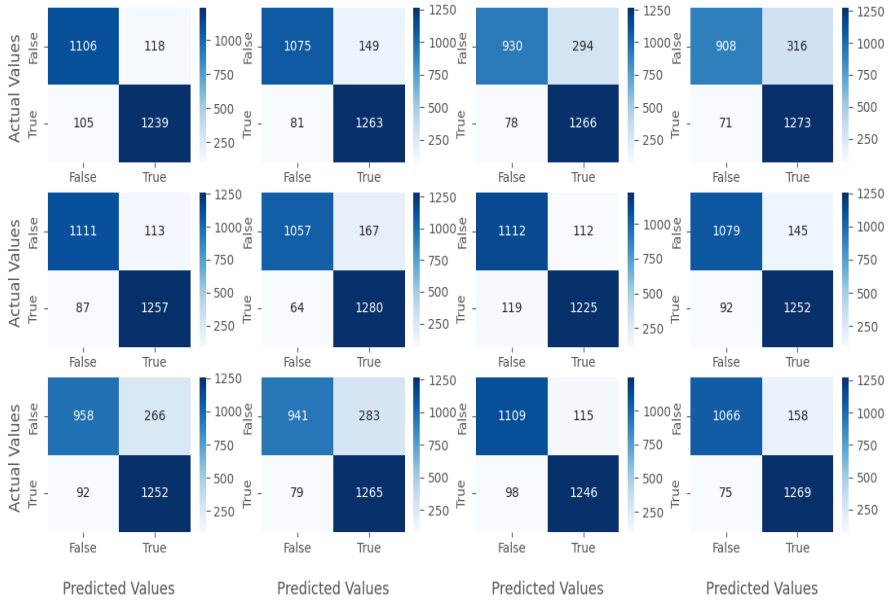
Σχήμα 4.29: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes Classifier στο Dataset 1



Σχήμα 4.30: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes Classifier στο Dataset 2



Σχήμα 4.31: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes Classifier στο Dataset 1



Σχήμα 4.32: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multinomial Naïve Bayes στο Dataset 2

4.1.9 Bernoulli Classifier

Στον Πίνακα 4.17 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Bernoulli Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf και count δίνει τα ίδια αποτελέσματα F1-score. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερα αποτελέσματα στο F1-score σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.89 F1-score, και ακολουθεί με τις ίδιες παραμέτρους και αποτέλεσμα το Tf-Idf. Στο Σχήμα 4.33 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.17. Αντίστοιχα, στο Σχήμα 4.35 βλέπουμε τους confusion matrix για το Dataset 1.

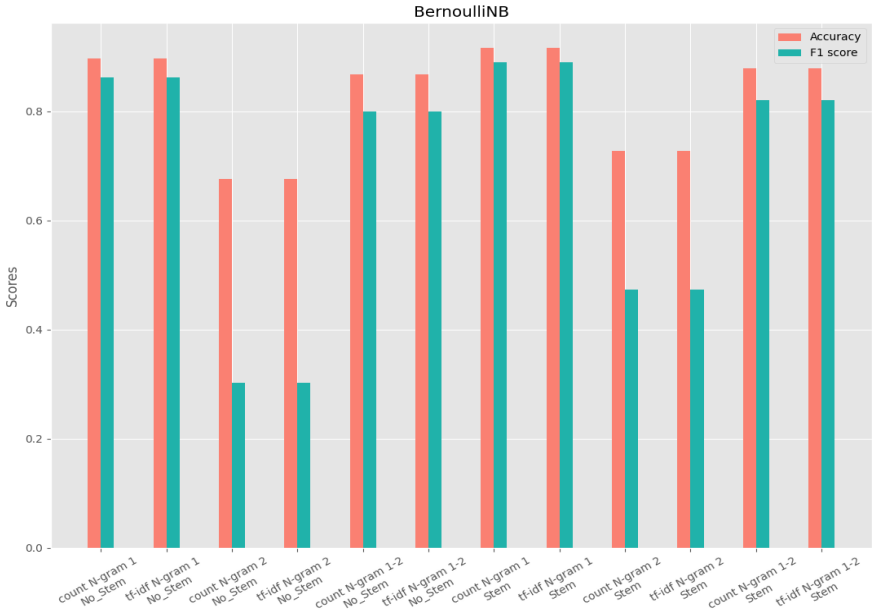
Στον Πίνακα 4.18 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Bernoulli classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf και count δίνει τα ίδια αποτελέσματα F1-score. Επίσης βλέπουμε ότι η χρήση stemming αυξάνει παντού τις επιδόσεις. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, με stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.92 F1-score, και ακολουθεί με τις ίδιες παραμέτρους και αποτέλεσμα το count. Στο Σχήμα 4.34 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.18. Αντίστοιχα, στο Σχήμα 4.36 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8965	0.8616
tf-idf	N-gram 1	No Stem	0.8965	0.8616
count	N-gram 2	No Stem	0.6758	0.3025
tf-idf	N-gram 2	No Stem	0.6758	0.3025
count	N-gram 1-2	No Stem	0.8672	0.8
tf-idf	N-gram 1-2	No Stem	0.8672	0.8
count	N-gram 1	Stem	0.916	0.8895
tf-idf	N-gram 1	Stem	0.916	0.8895
count	N-gram 2	Stem	0.7266	0.4737
tf-idf	N-gram 2	Stem	0.7266	0.4737
count	N-gram 1-2	Stem	0.8789	0.8208
tf-idf	N-gram 1-2	Stem	0.8789	0.8208

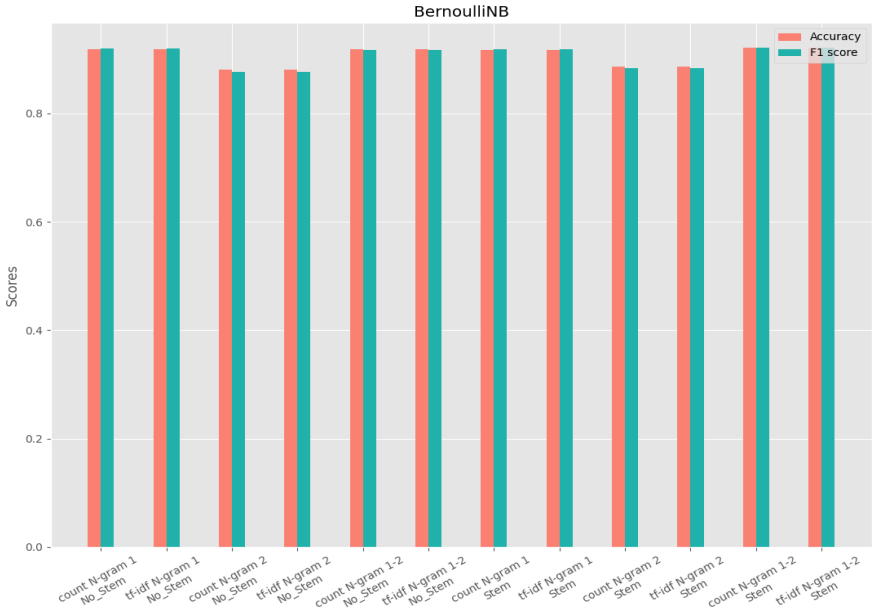
Πίνακας 4.17: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.9182	0.92
tf-idf	N-gram 1	No Stem	0.9182	0.92
count	N-gram 2	No Stem	0.8805	0.876
tf-idf	N-gram 2	No Stem	0.8805	0.876
count	N-gram 1-2	No Stem	0.9178	0.9175
tf-idf	N-gram 1-2	No Stem	0.9178	0.9175
count	N-gram 1	Stem	0.9167	0.9181
tf-idf	N-gram 1	Stem	0.9167	0.9181
count	N-gram 2	Stem	0.8867	0.8842
tf-idf	N-gram 2	Stem	0.8867	0.8842
count	N-gram 1-2	Stem	0.921	0.9212
tf-idf	N-gram 1-2	Stem	0.921	0.9212

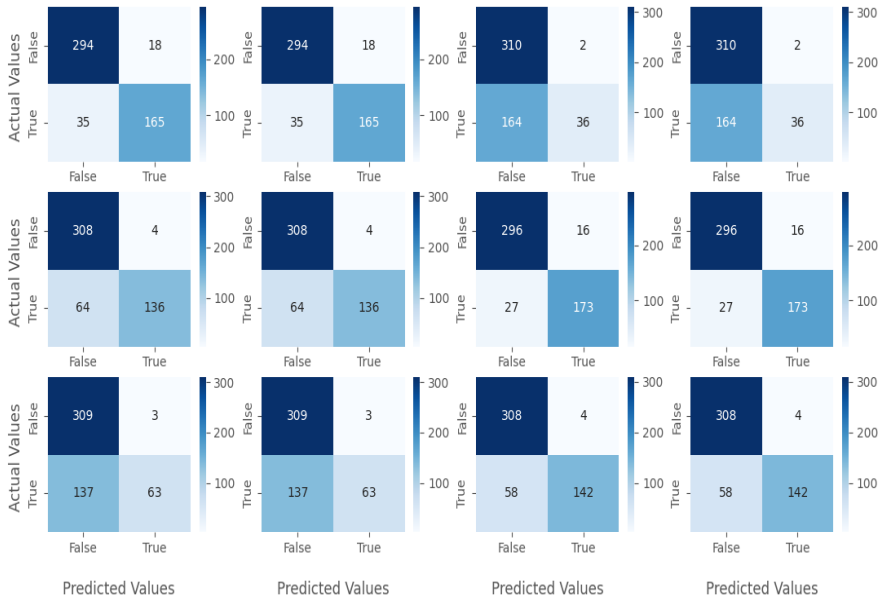
Πίνακας 4.18: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli Classifier στο Dataset 2



Σχήμα 4.33: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli Classifier στο Dataset 1



Σχήμα 4.34: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli Classifier στο Dataset 2



Σχήμα 4.35: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli Classifier στο Dataset 1

Σχήμα 4.36: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Bernoulli στο Dataset 2

4.1.10 Multilayer Perceptron Classifier

Στον Πίνακα 4.19 δείχνουμε τις τιμές accuracy και F1-score πέτυχε ο Multilayer Perceptron Classifier στο Dataset 1, με την χρήση stemming ή όχι στα δεδομένα αλλά και για διαφορετικές επιλογές N-gram. Παρατηρώντας τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα F1-score σε σχέση με τον μοντέλο count σχεδόν σε όλες τις περιπτώσεις. Επίσης φαίνεται ότι η χρήση stemming δίνει καλύτερο F1-score σχεδόν σε όλα τα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο count, με stemming και N-gram (1), οπότε και επιτεύχθηκε κοντά στο 0.85 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο το μοντέλο Tf-Idf, οπότε και επιτεύχθηκε κοντά στο 0.85 F1-score. Στο Σχήμα 4.37 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.19. Αντίστοιχα, στο Σχήμα 4.39 βλέπουμε τους confusion matrix για το Dataset 1.

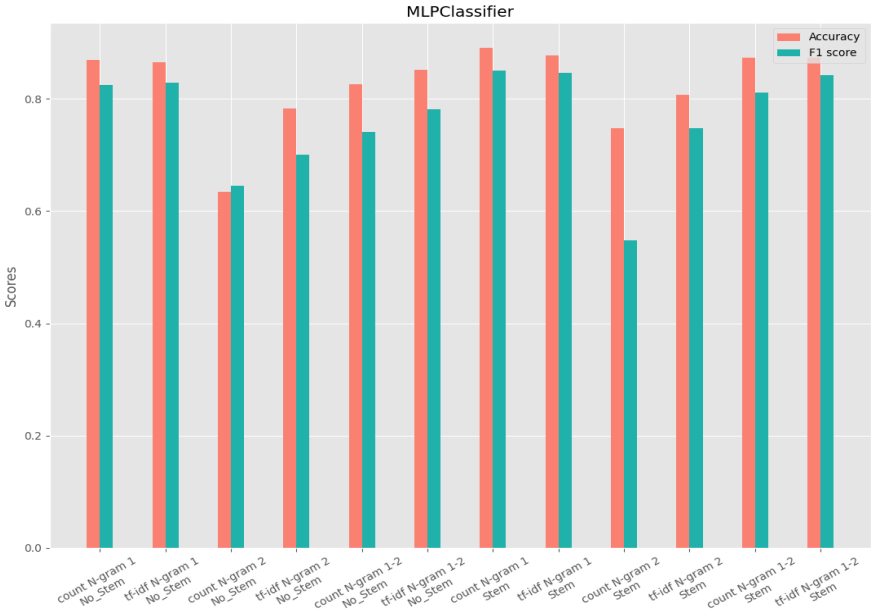
Στον Πίνακα 4.20 δείχνουμε τις τιμές accuracy και F1-score που πέτυχε ο Multilayer Perceptron Classifier στο Dataset 2. Όπως φαίνεται από τα αποτελέσματα, το μοντέλο Tf-Idf δίνει καλύτερα F1-score σε σχέση με το μοντέλο count σχεδόν σε όλες τις μετρήσεις. Επίσης βλέπουμε ότι η χρήση stemming δίνει χειρότερα αποτελέσματα. Η ψηλότερη τιμή F1-score παρατηρήθηκε όταν χρησιμοποιήσαμε το μοντέλο Tf-Idf, χωρίς stemming και N-gram (1, 2), οπότε και επιτεύχθηκε κοντά στο 0.94 F1-score, και ακολουθεί η δεύτερη καλύτερη όταν άλλαξε μόνο το μοντέλο count, οπότε και επιτεύχθηκε κοντά στο 0.93 F1-score. Στο Σχήμα 4.38 βλέπουμε την γραφική απεικόνιση σε Bar chart των μετρικών accuracy και F1-score που έχουμε στον Πίνακα 4.20. Αντίστοιχα, στο Σχήμα 4.40 βλέπουμε τους confusion matrix για το Dataset 2.

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.8691	0.8241
tf-idf	N-gram 1	No Stem	0.8652	0.8279
count	N-gram 2	No Stem	0.6348	0.6452
tf-idf	N-gram 2	No Stem	0.7832	0.7008
count	N-gram 1-2	No Stem	0.8262	0.7405
tf-idf	N-gram 1-2	No Stem	0.8516	0.7816
count	N-gram 1	Stem	0.8906	0.8495
tf-idf	N-gram 1	Stem	0.877	0.8467
count	N-gram 2	Stem	0.748	0.5474
tf-idf	N-gram 2	Stem	0.8066	0.7481
count	N-gram 1-2	Stem	0.873	0.8116
tf-idf	N-gram 1-2	Stem	0.873	0.8426

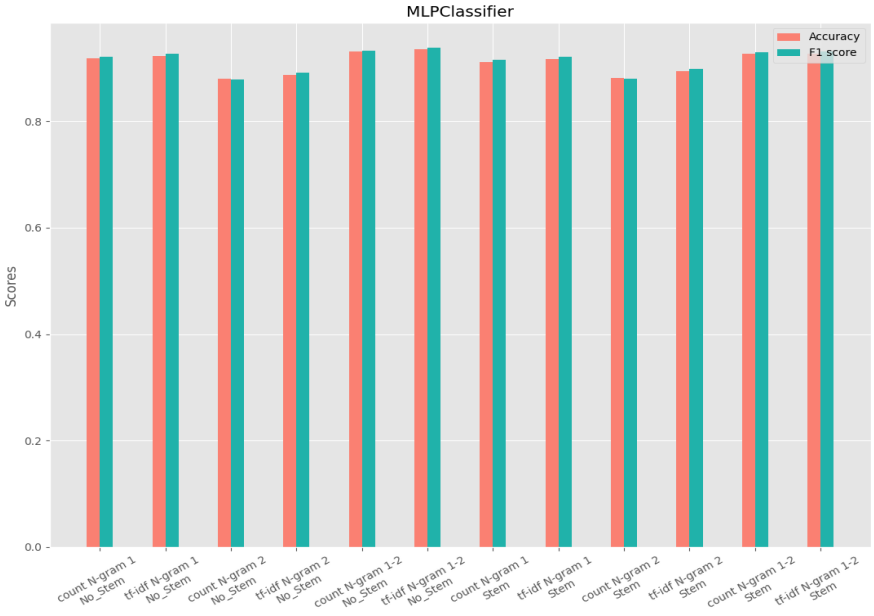
Πίνακας 4.19: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 1

Vectorizer	N-gram	Stemming	Accuracy	F1 score
count	N-gram 1	No Stem	0.9186	0.9219
tf-idf	N-gram 1	No Stem	0.9233	0.927
count	N-gram 2	No Stem	0.8801	0.8781
tf-idf	N-gram 2	No Stem	0.8875	0.8919
count	N-gram 1-2	No Stem	0.9311	0.9331
tf-idf	N-gram 1-2	No Stem	0.935	0.9383
count	N-gram 1	Stem	0.9108	0.9149
tf-idf	N-gram 1	Stem	0.9174	0.9218
count	N-gram 2	Stem	0.8816	0.88
tf-idf	N-gram 2	Stem	0.8937	0.8985
count	N-gram 1-2	Stem	0.9272	0.9292
tf-idf	N-gram 1-2	Stem	0.9272	0.931

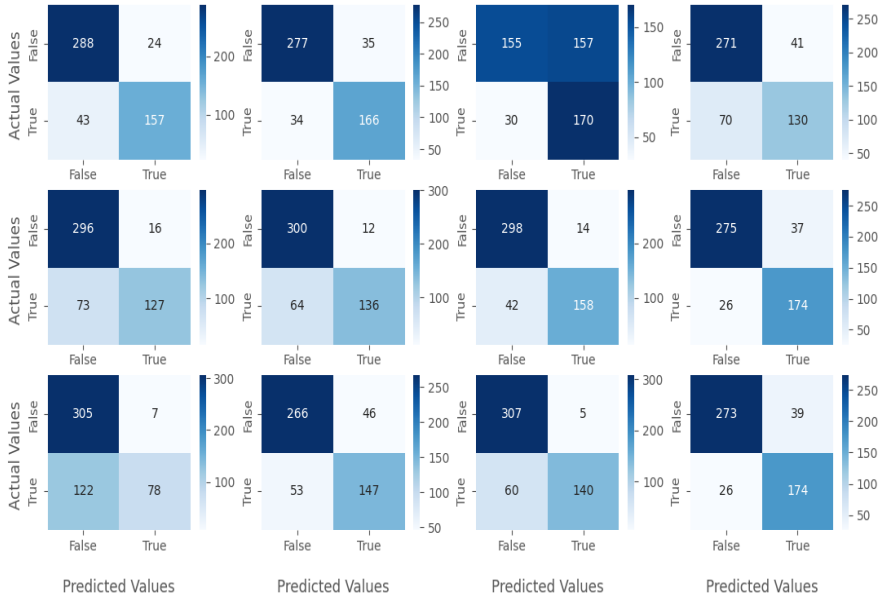
Πίνακας 4.20: Μετρικές για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 2



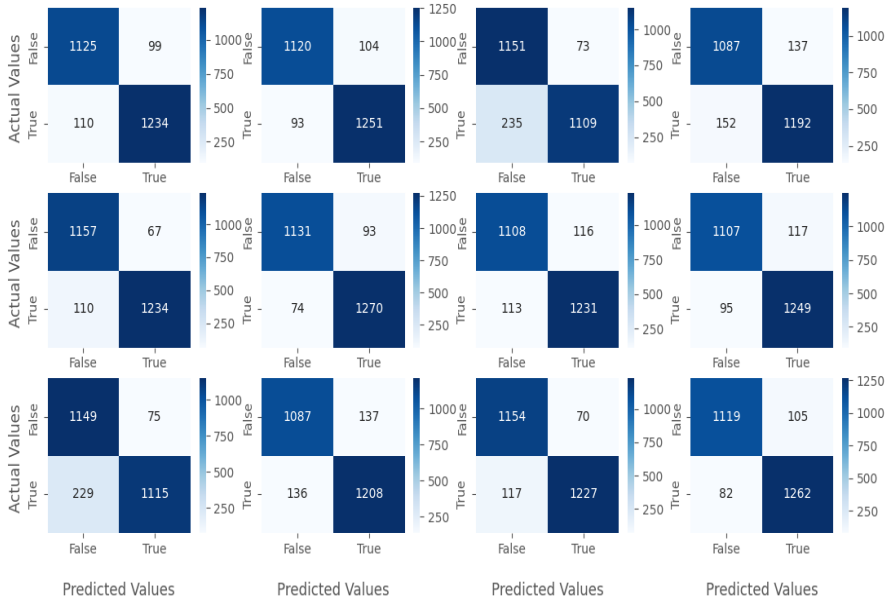
Σχήμα 4.37: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 1



Σχήμα 4.38: Bar chart των μετρικών Accuracy και F1-score για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 2



Σχήμα 4.39: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 1



Σχήμα 4.40: Confusion matrix για διαφορετικά μεγέθη λεξικού όταν χρησιμοποιήθηκε Multilayer Perceptron Classifier στο Dataset 2

4.2 Συμπεράσματα πειραματικής αξιολόγησης

Αξιολογώντας τα αποτελέσματα που πήραμε από τους αλγορίθμους που παρουσιάστηκαν στην προηγούμενη ενότητα καταλήγουμε στο συμπέρασμα ότι ο εντοπισμός ψευδών ειδήσεων με την χρήση μεθόδων Μηχανικής Μάθησης είναι εφικτός, και μάλιστα μπορεί να επιτευχθεί με αρκετά μεγάλη ακρίβεια. Στην παρούσα πτυχιακή, με τη χρήση δυο διαφορετικών συλλογών δεδομένων, φαίνεται άμεσα η σύγκριση μεταξύ 12 διαφορετικών τρόπων προ-επεξεργασίας και μετατροπής κειμένων σε διανύσματα και 10 αλγόριθμων κατηγοριοποίησης.

Στους Πίνακες 4.21, 4.22 έχουμε συγκεντρώσει τις μέγιστες τιμές F1-Score που πέτυχε ο κάθε αλγόριθμος στο κάθε dataset, ανάλογα με την προετοιμασία των δεδομένων που δέχτηκε ως είσοδο (δηλαδή την χρήση stemming, την επιλογή N-gram και vectorizer). Παρατηρώντας του πίνακες βλέπουμε για το πρώτο dataset αποδείχτηκε καλύτερος ο αλγόριθμος Bernoulli Naïve Bayes Classifier με F1-score = 0.8895 ενώ για το δεύτερο dataset ο καλύτερος αλγόριθμος είναι ο Multilayer Perceptron Classifier με F1-score = 0.9383. Ως γενικότερο συμπέρασμα μελετώντας όλα τα αποτελέσματα βλέπουμε η χρήση Tf-Idf είναι πιο αποδοτική από την χρήση count στις περισσότερες περιπτώσεις καθώς επίσης η χρήση stemming φαίνεται να βοηθά περισσότερο στην κατηγοριοποίηση των κειμένων ειδικά στο πρώτο dataset που περιέχει τα ελληνικά tweets. Τέλος το μοντέλο N-gram 1, N-gram 1,2 είναι τα καλύτερα και παρουσιάζουν ελάχιστες διαφορές στην απόδοση των αλγόριθμων.

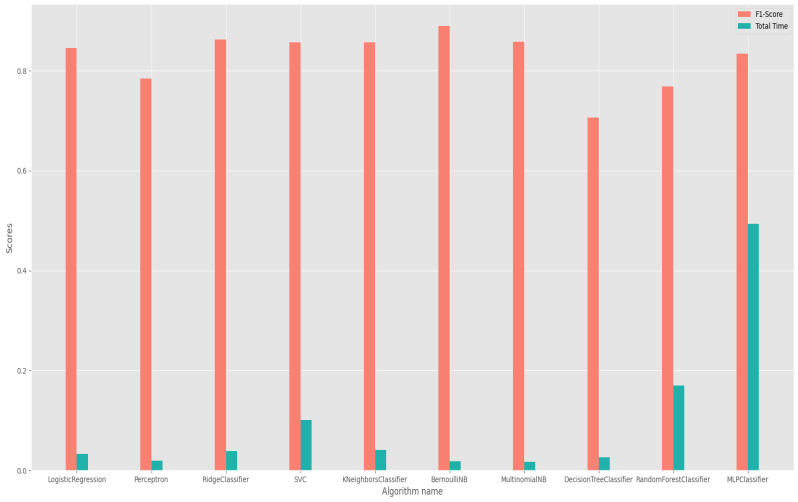
Πέρα από την σύγκριση των F1-score για την επιλογή του πιο κατάλληλου αλγορίθμου ίσως πρέπει να λάβουμε υπόψιν και τον χρόνο που απαιτεί να εκτελεστεί ο κάθε αλγόριθμος. Στα Σχήματα 4.41 4.42 φαίνονται οι διαφορετικοί χρόνοι εκτέλεσης του κάθε αλγορίθμου με χρήση stemming N-gram 1, και Tf-Idf Vectorizer για το Dataset 1 και το Dataset 2.

Algorithm	Vectorizer	N-gram	Stemming	F1-Score
Perceptron Classifier	count	N-gram 1-2	stemming	0.8159
Logistic Regression Classifier	count	N-gram 1-2	stemming	0.8579
Ridge Classifier	tf-idf	N-gram 1	stemming	0.8617
k-Nearest Neighbours Classifier	tf-idf	N-gram 1	stemming	0.8571
Decision Tree Classifier	count	N-gram 1-2	stemming	0.7136
Random Forest Classifier	tf-idf	N-gram 1	stemming	0.7692
Support Vector Machine Classifier	tf-idf	N-gram 1-2	stemming	0.8704
Multinomial Naive Bayes Classifier	tf-idf	N-gram 1	stemming	0.8579
Bernouli Classifier	tf-idf	N-gram 1	stemming	0.8895
Multilayer Perceptron Classifier	count	N-gram 1	stemming	0.8495

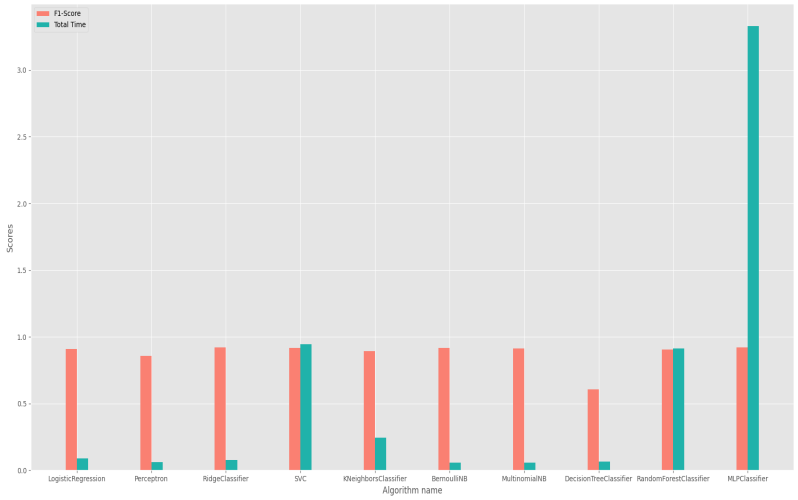
Πίνακας 4.21: Μεγαλύτερες τιμές F1-score για τον κάθε αλγόριθμο Μηχανικής Μάθησης στο Dataset 1

Algorithm	Vectorizer	N-gram	stemming	F1-Score
Perceptron Classifier	count	N-gram 1-2	stemming	0.8758
Logistic Regression Classifier	tf-idf	N-gram 1	stemming	0.9099
Ridge Classifier	tf-idf	N-gram 1-2	no stemming	0.9292
k-Nearest Neighbours Classifier	tf-idf	N-gram 1-2	no stemming	0.8980
Decision Tree Classifier	tf-idf	N-gram 2	stemming	0.767
Random Forest Classifier	count	N-gram 1	stemming	0.9066
Support Vector Machine Classifier	tf-idf	N-gram 1-2	no stemming	0.9299
Multinomial Naive Bayes Classifier	count	N-gram 1-2	no stemming	0.9263
Bernouli Classifier	tf-idf	N-gram 1-2	stemming	0.9212
Multilayer Perceptron Classifier	tf-idf	N-gram 1-2	no stemming	0.9383

Πίνακας 4.22: Μεγαλύτερες τιμές F1-score για τον κάθε αλγόριθμο Μηχανικής Μάθησης στο Dataset 2



Σχήμα 4.41: Bar chart των μετρικών F1-score και του συνολικού χρόνου εκτέλεσης σε sec για κάθε αλγόριθμο Μηχανικής Μάθησης στο Dataset 1



Σχήμα 4.42: Bar chart των μετρικών F1-score και του συνολικού χρόνου εκτέλεσης σε sec για κάθε αλγόριθμο Μηχανικής Μάθησης στο Dataset 2

Από τα γραφήματα παρατηρούμε ότι και στα 2 dataset ο Multilayer Perceptron Classifier είχε το μεγαλύτερο χρόνο εκτέλεσης καθώς επίσης ο Support Vector Machine Classifier και ο Random Forest Classifier είχαν αρκετά πιο υψηλό χρόνο εκτέλεσης σε σχέση με τους υπόλοιπους. Ο χρόνος εκτέλεσης στα πλαίσια της πτυχιακής δεν αποτέλεσε κριτήριο για επιλογή κάποιου αλγορίθμου εξαιτίας του περιορισμένου όγκου συνόλου δεδομένων, σαφώς όμως παίζει σημαντικό ρόλο σε κάποια μελέτη με τεράστιο όγκο δεδομένων. Τέλος ο κώδικας εκτελέστηκε σε περιβάλλον Windows 11, με επεξεργαστή Ryzen 5 5600x 3.7 GHz 6 πυρήνων στα 7nm της αρχιτεκτονικής Zen 3, 32GB διαθέσιμη RAM στα 3200 MHz και M2 SSD. Κατά την λειτουργία του κώδικα δεν εκτελούνταν άλλες εντολές από τον χρήστη, και το περιβάλλον εκτέλεσης του κώδικα ήταν το Visual Studio Code.

Κεφάλαιο 5

Επίλογος και Μελλοντικές Υλοποιήσεις

5.1 Επίλογος

Στην παρούσα πτυχιακή εργασία εντοπίσαμε τις ψευδείς ειδήσεις σχετικά με τον Covid-19 σε δυο συλλογές δεδομένων από το Twitter. Δοκιμάσαμε μια σειρά από τεχνικές Μηχανικής Μάθησης για την ταξινόμηση των tweets σε αληθείς και ψευδείς ειδήσεις. Όλοι οι αλγόριθμοι δοκιμάστηκαν σε δυο διαφορετικά σύνολα δεδομένων, σε ένα που περιλαμβάνει ελληνικά κι σε ένα με αγγλόφωνα tweets που αφορούν στον Covid-19. Η εκτενής πειραματική αξιολόγηση των αλγορίθμων αποδεικνύει ότι η γλώσσα των tweets δεν επηρεάζει καθόλου την ποιότητα των αποτελεσμάτων, επίσης αναδεικνύει τον Bernoulli Naïve Bayes Classifier ως τον πλέον κατάλληλο για ταξινόμηση δεδομένων που προέρχονται από το πρώτο dataset, ενώ για το δεύτερο dataset ήταν ο Multilayer Perceptron Classifier.

5.2 Μελλοντικές Υλοποιήσεις

Ως μελλοντική εργασία θα μπορούσαν να υλοποιηθούν μερικές απο τις ακόλουθες προτάσεις:

1. Δοκιμή περισσότερων Stemmers, όπως για παράδειγμα Porter Stemmer, και lemmatizers, όπως για παράδειγμα WordNetLemmatizer.

2. Δοκιμή των αλγόριθμων Doc2Vec [1], Bert [2] σαν μέθοδοι διανυσματοποιήσης.

3. Εφαρμογή περισσότερων αλγόριθμων κατηγοριοποίησης, όπως AdaBoost [3], RNN [4] ή LSTM [5] (Deep Learning) που υποστηρίζονται από το tensorflow [6].

4. Χρήση της βιβλιοθήκης Hyperopt-Sklearn [7] που αποτελεί ένα εργαλείο που εκτελεί μια ομάδα από τους αλγόριθμους που χρησιμοποιούνται στο Scikit-Learn. Το εργα-

[1] https://radimrehurek.com/gensim/models/doc2vec.html
[2] https://www.tensorflow.org/text/tutorials/classify$_{t}ext_{w}ith_{b}ert$
[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
[4] https://www.tensorflow.org/guide/keras/rnn
[5] https://www.tensorflow.org/api$_{d}ocs/python/tf/keras/layers/LSTM$
[6] https://www.tensorflow.org/resources/learn-ml
[7] https://hyperopt.github.io/hyperopt-sklearn/

λείο αυτό εκτελεί δοκιμές χρησιμοποιώντας τα δεδομένα εκπαίδευσης, εφαρμόζοντας διάφορους αλγόριθμους με διαφορετικές παραμέτρους, με σκοπό να καταλήξει στο κατάλληλο μοντέλο Μηχανικής Μάθησης που ανταποκρίνεται στην βέλτιστη λύση του προβλήματος.

Κεφάλαιο 6

Βιβλιογραφία

1. Emma Cueva, Grace Ee, Akshat Iyer, Alexandra Pereira, Alexander Roseman, Dayrene Martinez Detecting Fake News on Twitter Using Machine Learning Models New Jersey's Governor's School of Engineering and Technology July 24, 2020
2. Shai Shalev-Shwartz and Shai Ben-David. Understanding machine learning: From theory to algorithms. Cambridge university press (2014)
3. E. Alpaydin, Introduction to machine learning. MIT press, 2020.
4. Shagan Sah Machine Learning: A Review of Learning Types Shagan Sah Rochester Institute of Technology, Rochester Institute of Technology, July 2020.
5. Ευστάθιος Γ. Κύρκος, Επιχειρηματική Ευφυΐα Εξόρυξη Δεδομένων 2015.
6. Sebastian Raschka and Vahid Mirjalili Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition 2020
7. Padraig Cunningham and Sarah Jane Delany, k-Nearest neighbour classifiers.ACM Computing Surveys 54 April 2007
8. W. S. McCulloch and W. Pitts,(A Logical Calculus of the Ideas) Immanent in Nervous Activity Bulletin of Mathematical Biophysics, 5(4): 115-133, 1943
9. F. Rosenblatt, The Perceptron: A Perceiving and Recognizing Automaton,Cornell Aeronautical Laboratory, 1957
10. K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in Advances in neural information processing systems, pp. 1473–1480, 2006.
11. Ι. Βλαχάβας, Π. Κεφαλάς, Ν. Βασιλειάδης, Φ. Κόκκορας, Η. Σακελλαρίου ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ (Δ' ΕΚΔΟΣΗ), Νοεμβρίου 2020
12. Manning, C.D., Raghavan, P. and Schutze, H. Introduction to information retrieval (Vol. 39, pp. 1041-4347). Cambridge: Cambridge University Press 2008.
13. Wikipedia: Naive Bayes classifier

14. H. Taud and J. Mas, “Multilayer perceptron (mlp),” in Geomatic Approaches for Modeling Land Change Scenarios. Springer, pp. 451–455, 2018.

15. F. Rosenblatt, “Principles of neurodynamics. perceptrons and the theory of brain mechanisms,” Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

16. Durgesh Srivastava and Lekha Bhambhu Data classification using support vector machine.Article in Journal Theoretical and Applied Information Technology,February 2010.

17. Liddy, E.D., Natural Language Processing, In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc, 2001

18. Singh J. Gupta, V. Text stemming: Approaches, applications, and challenges. ACM Computing Surveys (CSUR), 49(3), 1-46, 2016.

19. Τρυφωνόπουλος Χρήστος Ανάκτηση και Εξόρυξη Πληροφοριών Προεπεξεργασία Πανεπιστήμιο Πελοποννήσου Τμήμα Πληροφορικής Τηλεπικοινωνιών

20. www.capitalone.com/tech/machine-learning/understanding-tf-idf

21. www.towardsdatascience.com/understanding-word-n-grams-and-n-gram-probability-in-natural-language-processing-9d9eef0fa058

22. K Subba Reddy and E. Srinivasa Reddy Detecting Spam Messages in Twitter Data by Machine learning Algorithms using Cross Validation International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-12, October 2019

23. Fabrıcio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgılio Almeida Detecting Spammers on Twitter Computer Science Department, Universidade Federal de Minas Gerais Belo Horizonte, Brazil 2010

24. De Wang, Danesh Irani and Calton Pu. A social spam detection framework. In proceedings of the Eight annual collaboration,Electronic messaging, AntiAbuse and Spam Conference (CEAS 2011),2011

25. Benjamin Markines, Ciro Cattuto and Filippo Menczer. Social Spam Detection. ACM-2009

26. Xueying Zhang, Xianghan Zheng. A Novel Method for Spammer Detection in Social Networks. IEEE-2015.

27. Hailu Xu, Weiqing Sun, Ahmad Javaid, Efficient Spam Detection across online social networks, IEEE-2015

28. Zellers, R., Holtzman, A., Rashkin, H., Bisk, Y., Farhadi, A., Roesner, F., Choi, Y. (2019). Defending against neural fake news. In Advances in Neural Information Processing Systems (pp. 9054-9065), 2019

29. https://backlinko.com/twitter-users

30. https://www.scielosp.org/pdf/rpsp/2021.v45/e56/en

31. https://el.wikipedia.org/wiki/

32. https://www.csc.com.gr/machine-learning-

33. Computing Machinery and Intelligence.Author(s): A. M. Turing.Source: Mind, New Series, Vol. 59, No. 236 , pp. 433-460.Published by: Oxford University Press on behalf of the Mind Association, Oct., 1950

34. Tom Dotz, Tom Hoobyar, Susan Sanders NLP: The Essential Guide to Neuro-Linguistic Programming 2013

35. Tom M. Mitchell Machine Learning ,McGraw-Hill March 1, 1997

36. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

37. Frank Rosenblatt Perceptron Simulation Experiments,Authors Cornell Aeronautical Laboratory, Inc., Buffalo, NY, USA, March 1960

38. Daniel Berrar, Cross-validation , January 2018

39. LEO BREIMAN Statistics Department,Random Forests University of California, Berkeley, CA 94720 ,Machine Learning, 45, 5–32, Kluwer Academic Publishers. Manufactured in The Netherlands, 2001

40. Siddhartha B S, An Interpretation of Lemmatization and Stemming in Natural Language Processing Article in Shanghai Ligong Daxue Xuebao/Journal of University of Shanghai for Science and Technology, January 2021

41. Ranjan Satapathy, Claudia Guerreiro, Iti Chaturvedi, Erik Cambria Phonetic-Based Microtext Normalization for Twitter Sentiment Analysis,,Nanyang Technological University,50 Nanyang Ave, 639798, Singapore, 2017 IEEE International Conference on Data Mining Workshops

Παράρτημα Α

Παράρτημα Α: Κώδικας

File preproccessing.py

```
import csv
from datetime import time
import emoji
import re
from greek_stemmer import GreekStemmer
from nltk.stem import PorterStemmer
from nltk.stem import SnowballStemmer
import time
from nltk.tokenize import word_tokenize
import string
from nltk.corpus import stopwords

import unicodedata as ud


ss = SnowballStemmer(language="english")
grstemer = GreekStemmer()
translator = str.maketrans(string.punctuation,' '*len(string.punctuation))
en_stops = stopwords.words('english')

el_stops = stopwords.words('greek')
for i in range(len(el_stops)):
    el_stops[i] = ''.join((c for c in ud.normalize('NFD', el_stops[i]) if ud.category(
                                                c) != 'Mn'))

en_stops = set(en_stops)
el_stops = set(el_stops)
stops  = en_stops.union(el_stops)

def strip_emoji(text):
    new_text = re.sub(emoji.get_emoji_regexp(), r" ", text)
    return new_text


def text_transform(s,en_stem,gr_stem):
    s  = s.lower()
    #remove urls
    s = re.sub(r'http\S+', '', s)
    #remove accents
```

```
    s = ''.join((c for c in ud.normalize('NFD', s) if ud.category(c) != 'Mn'))
    #remove punctuation marks
    s = re.sub(r'[^\w\s]',' ',s)
    #remove emojis
    s = strip_emoji(s)
    #remove \n and •
    s = s.replace('\n',' ')
    s = s.replace('•',' ')
    #remove digits
    s = ''.join(i for i in s if not i.isdigit())
    words = s.split()

    for i in range(len(words)):
        if words[i]  in stops :
                words[i]=' '
        if en_stem:
            words[i] = ss.stem(words[i])
        if gr_stem:
            words[i] =grstemer.stem(words[i].upper()).lower()

    s = " ".join(words)
    return ' '.join(s.split())

if __name__ == "__main__":

    with open('datafiles/data1.csv', 'r',encoding='utf-8') as infile, open('datafiles/
                                                readydata1nostem.csv', 'w',encoding='
                                                utf-8') as outfile:
        reader = csv.reader(infile)
        for row in reader:
            first = text_transform(row[0],False,False)
            second = row[10].lower()
            row_ = first+","+second+"\n"
            outfile.write(str(row_))
    with open('datafiles/data1.csv', 'r',encoding='utf-8') as infile, open('datafiles/
                                                readydata1stem.csv', 'w',encoding='utf-
                                                8') as outfile:
        reader = csv.reader(infile)
        for row in reader:
            first = text_transform(row[0],True,True)
            second = row[10].lower()
            row_ = first+","+second+"\n"
            outfile.write(str(row_))


    with open('datafiles/data2.csv', 'r',encoding='utf-8') as infile, open('datafiles/
                                                readydata2nostem.csv', 'w',encoding='
                                                utf-8') as outfile:
        reader = csv.reader(infile)
        for row in reader:
            first = text_transform(row[1],False,False)
            second = row[2]
            row_ = first+","+second+"\n"
            outfile.write(str(row_))

    with open('datafiles/data2.csv', 'r',encoding='utf-8') as infile, open('datafiles/
```

```
                                                readydata2stem.csv', 'w',encoding='utf-
                                                8') as outfile:
        reader = csv.reader(infile)
        for row in reader:
            first = text_transform(row[1],True,False)
            second = row[2]
            row_ = first+","+second+"\n"
            outfile.write(str(row_))
```

File dictionary.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import re
data1 = pd.read_csv('datafiles/readydata1nostem.csv')
data2 = pd.read_csv('datafiles/readydata1stem.csv')
data3 = pd.read_csv('datafiles/readydata2nostem.csv')
data4 = pd.read_csv('datafiles/readydata2stem.csv')
cv1 = TfidfVectorizer(ngram_range=[1,1])
cv2 = TfidfVectorizer(ngram_range=[1,1])
cv3 = TfidfVectorizer(ngram_range=[1,1])
cv4 = TfidfVectorizer(ngram_range=[1,1])
cv5 = TfidfVectorizer(ngram_range=[2,2])
cv6 = TfidfVectorizer(ngram_range=[2,2])
cv7 = TfidfVectorizer(ngram_range=[2,2])
cv8 = TfidfVectorizer(ngram_range=[2,2])
cv9 = TfidfVectorizer(ngram_range=[1,2])
cv10 = TfidfVectorizer(ngram_range=[1,2])
cv11 = TfidfVectorizer(ngram_range=[1,2])
cv12 = TfidfVectorizer(ngram_range=[1,2])
tf1 = cv1.fit_transform(data1['content'])
tf2 = cv2.fit_transform(data2['content'])
tf3 = cv3.fit_transform(data3['content'])
tf4 = cv4.fit_transform(data4['content'])
tf5 = cv5.fit_transform(data1['content'])
tf6 = cv6.fit_transform(data2['content'])
tf7 = cv7.fit_transform(data3['content'])
tf8 = cv8.fit_transform(data4['content'])
tf9 = cv9.fit_transform(data1['content'])
tf10 = cv10.fit_transform(data2['content'])
tf11 = cv11.fit_transform(data3['content'])
tf12 = cv12.fit_transform(data4['content'])
print("Data 1 No stem N-gram 1:"+str(tf1._shape[1]))
print("Data 1 Stem N-gram 1:"+str(tf2._shape[1]))
print("Data 2 No Stem N-gram 1:"+str(tf3._shape[1]))
print("Data 2 Stem N-gram 1:"+str(tf4._shape[1]))
print("Data 1 No stem N-gram 2:"+str(tf5._shape[1]))
print("Data 1 Stem N-gram 2:"+str(tf6._shape[1]))
print("Data 2 No Stem N-gram 1-2:"+str(tf7._shape[1]))
print("Data 2 Stem N-gram  1-2:"+str(tf8._shape[1]))
print("Data 1 No stem N-gram  1-2:"+str(tf9._shape[1]))
print("Data 1 Stem N-gram  1-2:"+str(tf10._shape[1]))
print("Data 2 No Stem N-gram  1-2:"+str(tf11._shape[1]))
print("Data 2 Stem N-gram  1-2:"+str(tf12._shape[1]))
```

File main.py

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from preprocessing import text_transform
import matplotlib.ticker as t
from sklearn import neighbors,metrics
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.pipeline import make_pipeline
import sys

d = {ord('\N{COMBINING ACUTE ACCENT}'):None}
np.set_printoptions(threshold=sys.maxsize)
data1 = pd.read_csv('datafiles/readydata1nostem.csv')
data2 = pd.read_csv('datafiles/readydata1stem.csv')
data3 = pd.read_csv('datafiles/readydata2nostem.csv')
data4 = pd.read_csv('datafiles/readydata2stem.csv')
label_mapping = {
    'fake':0,
    'true':1,
    'real':1
}
algorithms= [
                    LogisticRegression(solver='saga',penalty='elasticnet',l1_ratio=0 ,
                                                            class_weight='balanced')
                                                            ,
                    Perceptron(penalty="elasticnet",l1_ratio=1),
                    RidgeClassifier(solver='sag',tol=1e-3),
                    svm.SVC(kernel="sigmoid",cache_size=350  ),
                    KNeighborsClassifier(n_neighbors=14,weights='distance'),
                    BernoulliNB(alpha=0.1),
                    MultinomialNB(alpha=0.3),
                    tree.DecisionTreeClassifier(min_samples_split=0.5),
                    RandomForestClassifier() ,
                    MLPClassifier(hidden_layer_sizes=(10,)),
                ]
names = [
                     "count N-gram 1\nNo_Stem","tf-idf N-gram 1\nNo_Stem",
                     "count N-gram 2\nNo_Stem","tf-idf N-gram 2\nNo_Stem",
                     "count N-gram 1-2\nNo_Stem","tf-idf N-gram 1-2\nNo_Stem",
                     "count N-gram 1\nStem","tf-idf N-gram 1\nStem",
                     "count N-gram 2\nStem","tf-idf N-gram 2\nStem",
                     "count N-gram 1-2\nStem","tf-idf N-gram 1-2\nStem"
        ]
```

```
for algo in algorithms:
    accuracies_scores = []
    f1_scores = []
    precission_scores = []
    recall_scores=[]
    confusion_matrixs=[]
    for data in [data1,data2]:
        x = data['content'].values
        y = data['spam'].map(label_mapping).values
        x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=
                                                    1,stratify=y)
        models = []

        models.append(make_pipeline(CountVectorizer(ngram_range=[1,1]),algo))
        models.append(make_pipeline(TfidfVectorizer(ngram_range=[1,1]),algo))
        models.append(make_pipeline(CountVectorizer(ngram_range=[2,2]),algo))
        models.append(make_pipeline(TfidfVectorizer(ngram_range=[2,2]),algo))
        models.append(make_pipeline(CountVectorizer(ngram_range=[1,2]),algo))
        models.append(make_pipeline(TfidfVectorizer(ngram_range=[1,2]),algo))

        for model in models:
            #train model
            model.fit(x_train,y_train)
            prediction  = model.predict(x_test)
            accuracies_scores.append(round(metrics.accuracy_score(y_test,prediction),4
                                                        ))
            f1_scores.append(round(metrics.f1_score(y_test,prediction),4))
            confusion_matrixs.append(metrics.confusion_matrix(y_test,prediction))
    lis = ["count,N-gram 1,No Stem","tf-idf,N-gram 1,No Stem",
           "count,N-gram 2,No Stem","tf-idf,N-gram 2,No Stem",
           "count,N-gram 1-2,No Stem","tf-idf,N-gram 1-2,No Stem",
           "count,N-gram 1,Stem","tf-idf,N-gram 1,Stem",
           "count,N-gram 2,Stem","tf-idf,N-gram 2,Stem",
           "count,N-gram 1-2,Stem","tf-idf,N-gram 1-2,Stem"]
    with  open('results/'+type(algo).__name__[0:5].lower()+'_1.csv', 'w') as f:
        f.write("Vectorizer"+","+"N-gram"+","+"Stemming"+","+"Accuracy"+","+"F1 score\
                                                    n")
        for i ,j,z in zip(accuracies_scores,f1_scores,lis):
            f.write(str(z))
            f.write(",")
            f.write(str(i))
            f.write(",")
            f.write(str(j))
            f.write("\n")
        f.close()
    ind = np.arange(12)
    width = 0.10
    plt.figure(figsize=(13,9))
    axes = plt.axes()
    plt.style.use('ggplot')
    plt.bar(ind/2, accuracies_scores, width,color='salmon',label ='Accuracy')
    plt.bar(ind/2 + width , f1_scores,width, color='lightseagreen',label = 'F1 score')
    plt.title(type(algo).__name__)
    plt.ylabel("Scores")
    plt.xticks(ind/2+width/2,names)
```

```
    plt.legend(loc='best')
    plt.setp(axes.get_xticklabels(),rotation=30, horizontalalignment='center')


    plt.savefig('eikones/d'+type(algo).__name__.lower()[0:5]+"_1")
    plt.figure(figsize=(13,7))
    for index,i in enumerate(confusion_matrixs):
        plt.subplot(3, 4, index+1)
        ax = sns.heatmap(i, annot=True,cmap='Blues',fmt='g')
        if index ==0 or index ==4 or index == 8:
            ax.set_ylabel('Actual Values ');
        ax.set_xlabel('\nPredicted Values')


        ## Ticket labels - List must be in alphabetical order
        ax.xaxis.set_ticklabels(['False','True'])
        ax.yaxis.set_ticklabels(['False','True'])
    plt.savefig('eikones/f'+type(algo).__name__.lower()[0:5]+"_1")
```

File time.py

```
from os import times
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, Perceptron
from sklearn import svm
from sklearn.neighbors import  KNeighborsClassifier, NearestCentroid
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn import tree
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.pipeline import make_pipeline
import sys
import time


d = {ord('\N{COMBINING ACUTE ACCENT}'):None}
np.set_printoptions(threshold=sys.maxsize)
data = pd.read_csv('datafiles/readydata1stem.csv')


label_mapping = {
     'fake':0,
    'true':1,
    'real':1
}

x = data['content'].values
y = data['spam'].map(label_mapping).values
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.4,random_state=1,
                                        stratify=y)
algorithms= [
                    LogisticRegression(solver='saga',penalty='elasticnet',l1_ratio=0 ,
                                                            class_weight='balanced')
                                                            ,
                    Perceptron(penalty="elasticnet",l1_ratio=1),
                    RidgeClassifier(solver='sag',tol=1e-3),
                    svm.SVC(kernel="sigmoid",cache_size=350  ),
                    KNeighborsClassifier(n_neighbors=14,weights='distance'),
                    BernoulliNB(alpha=0.1),
                    MultinomialNB(alpha=0.3),
                    tree.DecisionTreeClassifier(min_samples_split=0.5),
                    RandomForestClassifier() ,
                    MLPClassifier(hidden_layer_sizes=(10,)),
                ]


models = []
for algo in algorithms:
    models.append(make_pipeline(TfidfVectorizer(),algo))
f1_scores = []
names = [type(algo).__name__ for algo in algorithms]
times=[]
for model,algo in zip(models,algorithms):

    t0= time.time()
    model.fit(x_train,y_train)

    prediction  = model.predict(x_test)
    t1 = time.time() - t0
    times.append(t1)
    f1_scores.append(metrics.f1_score(y_test,prediction))


ind = np.arange(len(algorithms))
width = 0.15
axes = plt.axes()
plt.style.use('ggplot')
plt.bar(ind, f1_scores, width,color='salmon',label ='F1-Score')
plt.bar(ind + width , times,width, color='lightseagreen',label = 'Total Time')
plt.xlabel("Algorithm name")
plt.ylabel("Scores")
plt.xticks(ind+ width/2 , names)
plt.legend(loc='best')
plt.setp(axes.get_xticklabels(),rotation=30, horizontalalignment='center')
plt.show()
```