ΥΣ19 Artificial Intelligence II

Πρώτη Εργασία.

Ονοματεπώνυμο: Απόστολος Καρβέλας

Α.Μ.: 1115201800312

Δεύτερο υποερώτημα (το πρώτο βρίσκεται στο τέλος):

ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ

Πρώτα πρέπει τρέξουμε το πρώτο κελί το οποίο περιέχει όλα τα modules και συναρτήσεις που χρειάζεται το πρόγραμμα. Στην συνέχεια μπορούμε να τρέξουμε κάθε κελί ξεχωριστά που αναπαριστά κάθε δοκιμή. Στο τέλος είναι και το τελικό μοντέλο μας.

Στο πάνω μέρος κάθε κελιού υπάρχουν 2 read_csv το οποίο διαβάζει το train και το validation set, οπότε για να τρέξει το πρόγραμμα με διαφορετικό testing dataset πρέπει να αλλάξει το path στο δεύτερο read_csv που το τοποθετεί στην μεταβλητή X_val. Τα αρχεία στο Colab notebook έχουν δοθεί μέσω των files στην αριστερή μεριά του collab, οπότε αν τα αρχεία βρίσκονται στον driver χρειάζεται να γίνει πρώτα mount.

ΕΙΣΑΓΩΓΗ

Στην εργασία υλοποιούμε έναν sentiment classifier για εμβόλια χρησιμοποιώντας softmax regression. Σταδιακά:

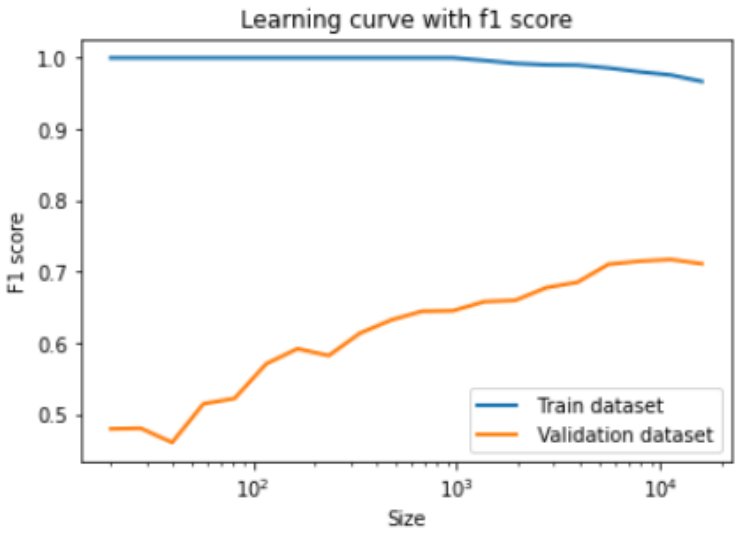
1. Εκτελούμε pre-processing στα δεδομένα αφαιρώντας τις άχρηστες πληροφορίες-λέξεις και προσθέτοντας prefix NOT_ στις λέξεις που εμφανίζονται μετά από negation. Οι δοκιμές γίνονται με απλό CountVectorizer και LogisticRegression χωρίς κάποιο ιδιαίτερο όρισμα γιατί θα πειραματιστούμε με αυτά στην συνέχεια.
2. Δοκιμάζουμε τους 3 βασικούς vectorizers: Countvectorizer, Tfidfvectorizer και Hashingvectorizer. Ταυτόχρονα θα πειράξουμε τις παραμέτρους με σκοπό να βελτιστοποιήσουμε τα metrics: f1 score, precision και recall και μέσω της learning curve να καταλάβουμε αν συμβαίνει overfitting ή underfitting. Οι μετρήσεις γίνονται με απλό LogisticRegressing.
3. Θα χρησιμοποιήσουμε την συνάρτηση LogisticRegressing της sklearn με παράμετρο multinomial για να υλοποιήσουμε softmax regression και θα πειραματιστούμε με τις παραμέτρους για να βρούμε το καλύτερο δυνατό score.

Για τον πειραματισμό και τις δοκιμές συμβουλευόμαστε κυρίως το f1 score

Data Pre-processing:

Δοκιμάζοντας τον sentiment classifier χωρίς καθόλου προεπεξεργασία δεδομένων υπολογίζουμε ότι ο vectorizer, σε αυτήν την περίπτωση απλός CountVectorizer, μαθαίνει το λεξιλόγιο για 32656 διαφορετικές λέξεις-features με την συνάρτησης fit. Μέσω της learning curve παρατηρούμε προφανή απόκλιση των 2 γραμμών train και validation ακόμα και για ολόκληρο το dataframe (train 0.966 ενώ το validation μόνο 0.712).

Train	0.966
Validation	0.712



Από το φαινόμενο αυτό συμπεραίνουμε ότι στον classifier γίνεται αρκετό overfitting.

Οπότε η αρχική ενέργεια για την λύση του προβλήματος είναι η προεπεξεργασία των δεδομένων. Σκοπός της επεξεργασίας είναι να αφαιρέσουμε όσες περισσότερες λέξεις-σύμβολα μπορούμε, οι οποίες δεν βοηθάνε το πρόγραμμα να κατηγοριοποιήσει τα δεδομένα, και από τα 2 datasets train και validation-test.

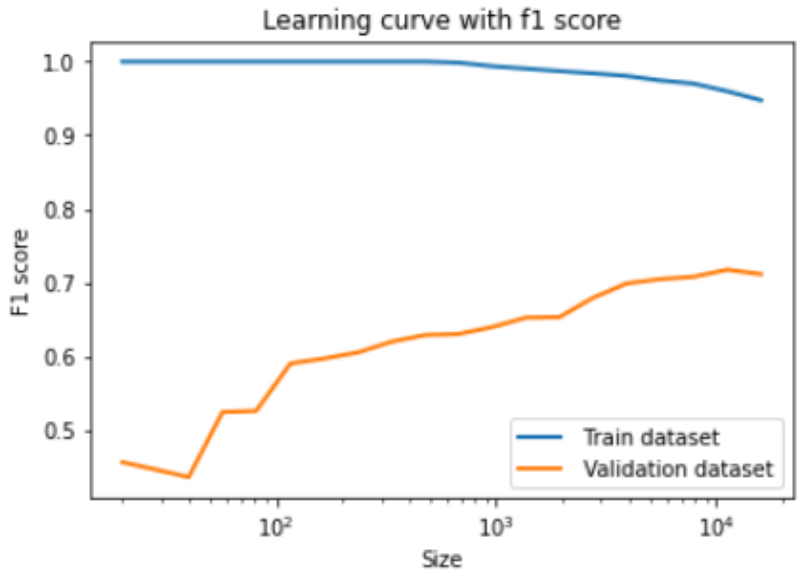
Άρα, αφαιρούμε τα:

- Whitespaces μέσω την συνάρτηση strip()
- Links και URLs
- Emojis και σύμβολα
- Σημεία στίξης
- Accents με το module unicodedata
- νούμερα

και μετατρέπουμε όλα τα γράμματα σε μικρά. Τέλος, για το καλύτερο sentiment analysis θα προσθέσουμε ένα prefix “NOT_” στις λέξεις που βρίσκονται μπροστά από token με logical negation όπως “n’t”, “not”, “no“ ή “never” ώστε ο classifier να μπορεί πιο εύκολα να καταλάβει αρνήσεις.

Με αυτές τις πράξεις μειώνονται οι διαφορετικές λέξεις σε 26254 ενώ ταυτόχρονα θα είναι και πιο εύκολο στον classifier να κατηγοριοποιήσει καθώς αφαιρέσαμε τις λέξεις που δεν αναλογούν σε κάποιο αίσθημα.

Επιπλέον αφαιρούμε τα stop words μέσω του vectorizer προσθέτοντας σαν παράμετρο του stop_words να ισούται με “english” οπότε κατεβαίνουν σε 25967 λέξεις.



Train	0.947
Validation	0.711

Παρατηρούμε ότι συνεχίζει να γίνεται overfitting και ας μειώνεται λίγο από pre-processing (0.947 f1 score του train και 0.711 στο validation) αυτό είναι λογικό καθώς και πάλι έχουμε υπερβολικά πολλά features.

Στην συνέχεια θα δοκιμάσουμε να κάνουμε stemming και lemmatization τα δεδομένα με την χρήση του module nltk.

Οπότε πρώτα κάνουμε split τα δεδομένα, καλούμε τις συναρτήσεις stem και lem οι οποίες εκτελούν την ανάλογη συνάρτηση και τα ξαναενώνουμε με την εντολή stringify. Οπότε με χρήση προεπεξεργασίας και stemming καταλήγουμε σε 21617 λέξεις (0.926 f1 score του train και 0.705 στο validation), βέβαια μπορεί να υπάρχει overstemming το οποίο μειώνει το τελικό precision, οπότε θα συμβουλευτούμε και τις μετρήσεις όπως το f1 score.

Train	0.926
Validation	0.705

Με lemmatization καταλήγει σε 24331 με καλύτερα scores (0.941 f1 score του train και 0.712 στο validation), οπότε θα προτιμήσουμε αυτό. Και με τις 2 τεχνικές stemming και lemmatization δεν αλλάζει το learning curve και συνεχίζουμε να έχουμε overfitting καθώς αφαιρέθηκαν πολύ λίγες λέξεις.

Train	0.941
Validation	0.712

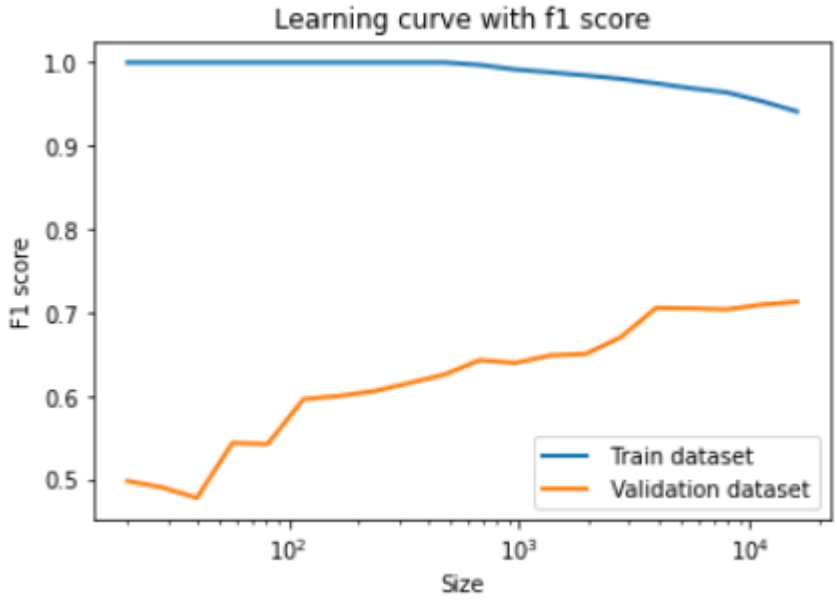
Vectorizers

CountVectorizer:

Στην αρχή θα δοκιμάσουμε να μετατρέψουμε τα δεδομένα σε μορφή που καταλαβαίνει η LogisticRegression μέσω Countvectorizer ο οποίος απλά μετράει τις φορές που εμφανίζεται μια λέξη και χρησιμοποιεί την τιμή αυτή σαν βάρος.

Αφού έχουμε κάνει preprocessing τα δεδομένα χρησιμοποιούμε τις παραμέτρους stop_words = 'english' για να καθαρίσουμε τα κείμενα από stop words. Με την χρήση του LogisticRegression παρατηρούμε πάλι μεγάλη απόκλιση των f1 score για το train και validation set (0.941 f1 score του train και 0.712 στο validation).

Train	0.941
Validation	0.712



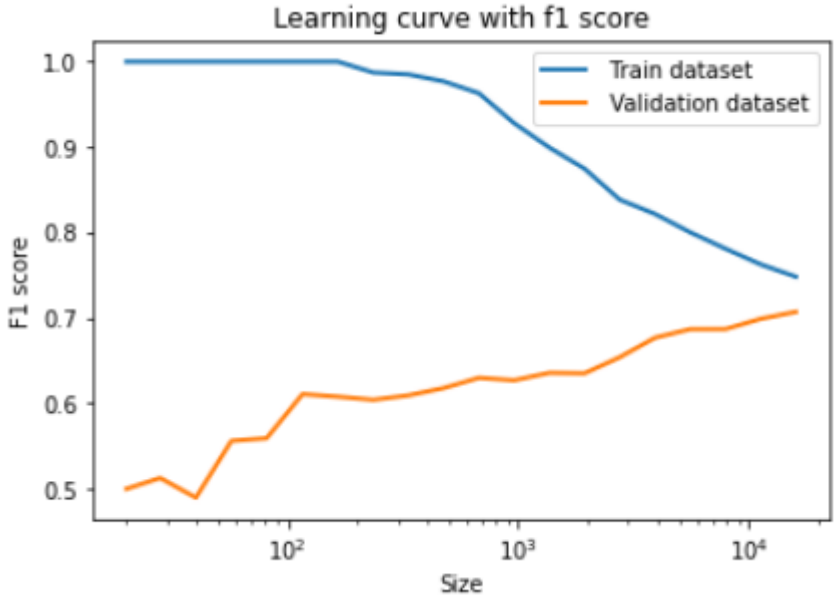
Οπότε συνεχίζουμε να έχουμε πρόβλημα overfitting. Το πιο σημαντικό πρόβλημα με την χρήση count vectorizer είναι ότι δίνει πολύ βάρος στις λέξεις που εμφανίζονται συχνά, που συνήθως είναι punctuation και stop words. Όμως στην δικιά μας περίπτωση έχουμε αφαιρέσει τις περισσότερες από αυτές τις λέξεις οπότε έχει μικρότερο περιθώριο λάθους. Εξετάζοντας το λεξιλόγιο, παρατηρούμε ότι λέξεις που εμφανίζονται το περισσότερο, οπότε έχουν και πολύ μεγαλύτερο βάρος, είναι σχετικές με την κατηγοριοποίηση των κειμένων

(π.χ. 'vaccine': 7055, 'measles': 4357, 'covid': 1898, 'kid': 1869, 'child': 1739, 'health': 1596).

Οπότε δεν χρειάζεται να χρησιμοποιήσουμε την παράμετρο max_df. Όμως καθώς κατεβαίνουμε στις τιμές-εμφανίσεις των λέξεων παρατηρούμε ότι οι λέξεις δεν έχουν κάποια σχέση με τα εμβόλια οπότε πρέπει να αλλάξουμε την παράμετρο min_df σε κάποιο νούμερο ώστε να μην περιλαμβάνει τις λέξεις αυτές, πειραματίζοντας με διάφορες τιμές του βλέπουμε ότι όσο περνάνε λιγότερες λέξεις στο λεξιλόγιο έχει λιγότερο overfitting αλλά

και μικρότερο f1 score. Μια ισορροπημένη αναλογία είναι min_df = 25 όπου (0.748 f1 score του train και 0.706 στο validation) με 982 λέξεις.

Train	0.748
Validation	0.706

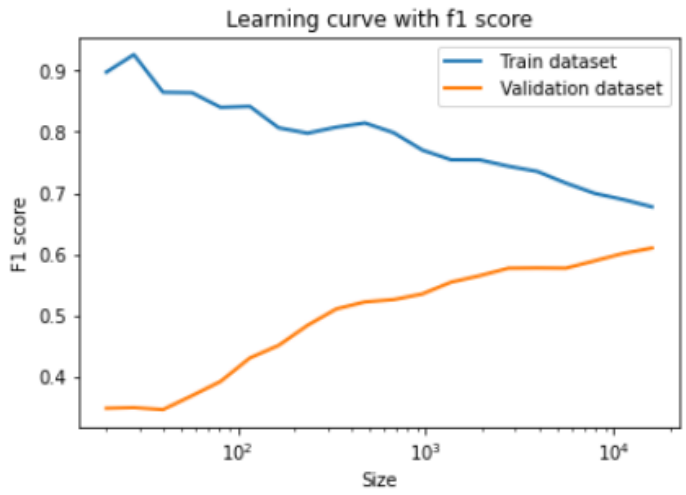


Από το learning curve δεν παρατηρούμε ούτε overfitting ούτε underfitting οπότε για τώρα είναι ένα καλό μοντέλο.

Τώρα θα δοκιμάσουμε να χρησιμοποιήσουμε bigrams το οποίο δέχεται σαν feature 2 συνεχόμενες λέξεις από το κείμενο. Αν ακολουθήσουμε την προηγούμενη τεχνική και βάλουμε στην παράμετρο min_df = 25 παρατηρούμε ότι δέχεται πολύ λίγα features καθώς στην γραπτή γλώσσα υπάρχουν πολλοί διαφορετικοί τρόποι για να πεις την ίδια πρόταση και πιο συγκεκριμένα μόνο 216 features. Αν όμως μειώσουμε το min_df = 7 έχουμε 1447 features οπότε:

Train f1 score 0.677 και validation f1 score 0.610 με learning curve:

Train	0.677
Validation	0.610

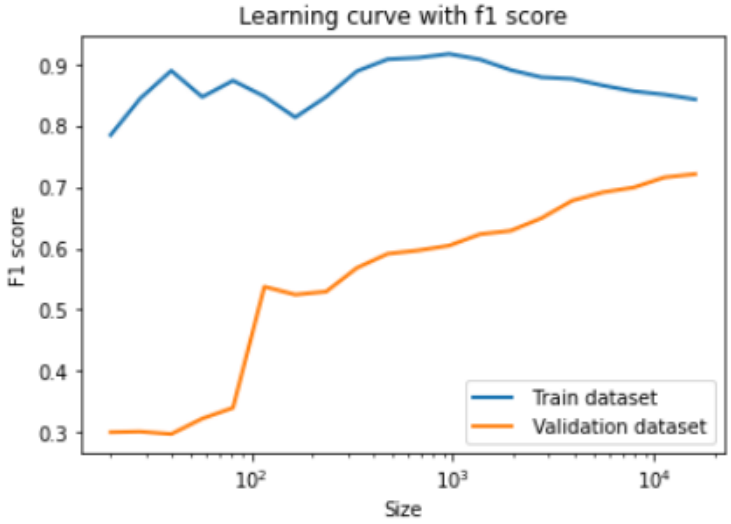


Παρατηρούμε ότι έχουν αρκετή απόκλιση μεταξύ τους και αρκετά χαμηλά f1 score οπότε δεν θα χρησιμοποιήσουμε bigrams.

TfidfVectorizer:

Ο TfidfVectorizer είναι θεωρητικά καλύτερος από τον Countvectorizer καθώς αφού υπολογίσει τις φορές που εμφανίζεται κάθε λέξη στα κείμενα, εκτελεί και τον αλγόριθμο tfidf για normalization των δεδομένων. Οπότε ο vectorizer δεν βγάζει αποτέλεσμα μόνο για την συχνότητα των λέξεων αλλά παρέχει και βάρος για κάθε μια από αυτή. Αυτό έχει σαν αποτέλεσμα οι λέξεις που εμφανίζονται πολλές φορές να μην έχουν τόσο μεγάλη διαφορά βάρους όπως γινόταν στον countvectorizer και έτσι το πρόγραμμα θα μπορεί πιο εύκολα να βγάλει συμπέρασμα, αφού θα βλέπει όλες τις λέξεις.

Τρέχουμε το πρόγραμμα με preprocessed data και TfidfVectorizer με παραμέτρους μόνο τα strop_words = “english”.

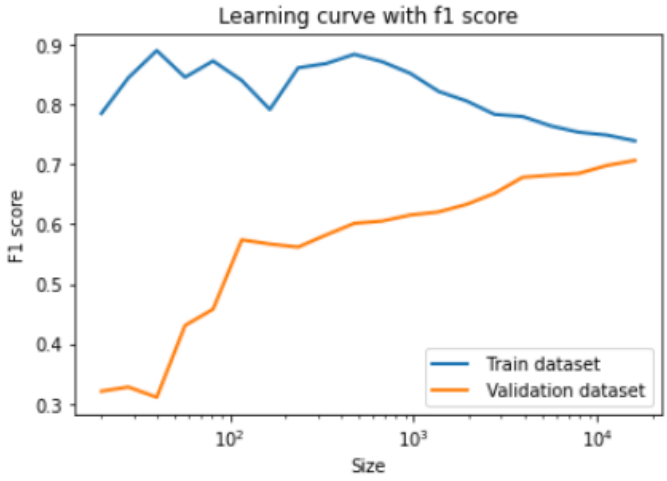


Train	0.842
Validation	0.721

Παρατηρούμε ότι ενώ και πάλι γίνεται overfitting, δεν υπάρχει τόσο μεγάλη απόκλιση των τιμών f1 score (αυτή την φορά 0.842 και 0.721) από εκείνα του Counvectorizer.

Οπότε θα ακολουθήσουμε την ίδια στρατηγική με τον Countvectorizer, δηλαδή θα πειράξουμε τον min_df σε ένα νούμερο ώστε να μην υπάρχει μεγάλη απόκλιση αλλά ταυτόχρονα να υπάρχουν κάποιες λέξεις στο λεξιλόγιο ώστε να αποφύγουμε underfitting. Αυξάνοντας το min_df κατά 5 σε κάθε εκτέλεση συμπεραίνουμε ότι ο Tfidfvectorizer συγκλίνει τις 2 τιμές f1 score, για train και validation, πιο γρήγορα από τον Countvectorizer (δηλαδή για μικρότερα min_df συγκλίνει πιο γρήγορα) και για min_df = 25 το Tfidfvectorizer συνεχίζει να είναι ελαφρώς καλύτερο από το Countvectorizer. Σχεδόν τα ίδια νούμερα μπορούμε να υπολογίσουμε και με min_df=0.0015. Πιο συγκεκριμένα έχουμε 0.739 f1 score του train και 0.705 στο validation.

Train	0.739
Validation	0.705



Και δεν γίνεται underfitting ή overfitting.

Συμπεραίνουμε ότι ο Tfidfvectorizer δουλεύει αρκετά καλύτερα για πολλά features ενώ για λιγότερα συνεχίζει να είναι ξεπερνάει τον Countervectorizer. Οπότε θα θεωρήσουμε μέχρι στιγμής την χρήση του Tfidfvectorizer σαν την καλύτερη τεχνική.

Hashingvectorizer:

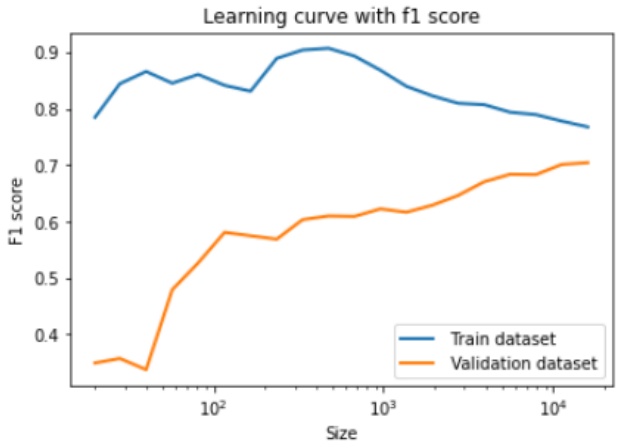
Τώρα θα συγκρίνουμε τον Hashingvectorizer με το καλύτερο μέχρι τώρα μοντέλο, δηλαδή τον Tfidfvectorizer. Από παραμέτρους θα βάλουμε μόνο τα stop_words = “english” και τον αριθμό των features που θα εισάγουμε στον vectorizer. Με n_features = 2**12 και n_features = 2**10 και στους 2 vectorizers (στον Tfidfvectorizer με την παράμετρο max_features) παρατηρούμε τα εξής αποτελέσματα:

Στον Tfidfvectorizer για 2**12 έχουμε f1 score για train 0.796 ενώ για validation 0.719

Train	0.796
Validation	0.719

Στον Hashingvectorizer για 2**12 έχουμε f1 score για train 0.767 ενώ για validation 0.704

Train	0.767
Validation	0.704



Παρατηρούμε ότι ενώ το f1 score του Tfidfvectorizer για το validation είναι μεγαλύτερο με του Hashing, έχει αρκετά πιο υψηλή απόκλιση μεταξύ του train. Οπότε για μεγάλα δεδομένα κάνει και λιγότερο overfitting.

Ενώ για features = 2**10 έχουμε:

Στον Tfidfvectorizer έχει f1 score στο train 0.738 ενώ για validation 0.702

Train	0.738
Validation	0.702

Στον Hashingvectorizer έχει f1 score στο train 0.709 και για validation 0.669

Train	0.709
Validation	0.669

Βλέπουμε ότι για λιγότερα features η tfidfvectorizer έχει και πιο μικρή απόκλιση των f1 score όπως και υψηλότερα από το hashingvectorizer. Οπότε συμπεραίνουμε ότι για μικρά δεδομένα όπως στο παράδειγμα αυτό ο tfidfvectorizer είναι καλύτερος οπότε και τον κρατάμε.

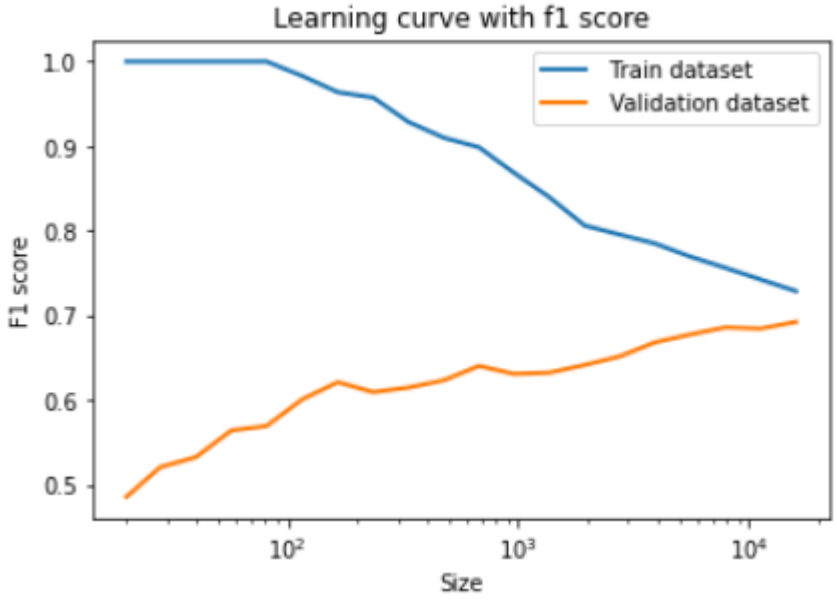
LogisticRegression

Για την υλοποίηση Softmax Regression θα χρησιμοποιήσουμε την συνάρτηση LogisticRegression της sklearn με multi_class = “multinomial”. Στην συνέχεια, δοκιμάζοντας διαφορετικούς solvers που χειρίζονται multinomial (‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’) δεν βλέπουμε κάποια ιδιαίτερη διαφορά στα δεδομένα οπότε θα κρατήσουμε τον default που είναι ο lbfgs.

Στην παράμετρο class_weight υπάρχει η επιλογή “balanced“ η οποία χρησιμοποιεί τις τιμές του label για να προσαρμόσει τα βάρη αντίθετα με την συχνότητα των λέξεων οπότε τρέχοντας το πρόγραμμα βγάζουμε:

Train f1 score = 0.729 και validation f1 score = 0.692 με learning curve:

Train	0.729
Validation	0.692

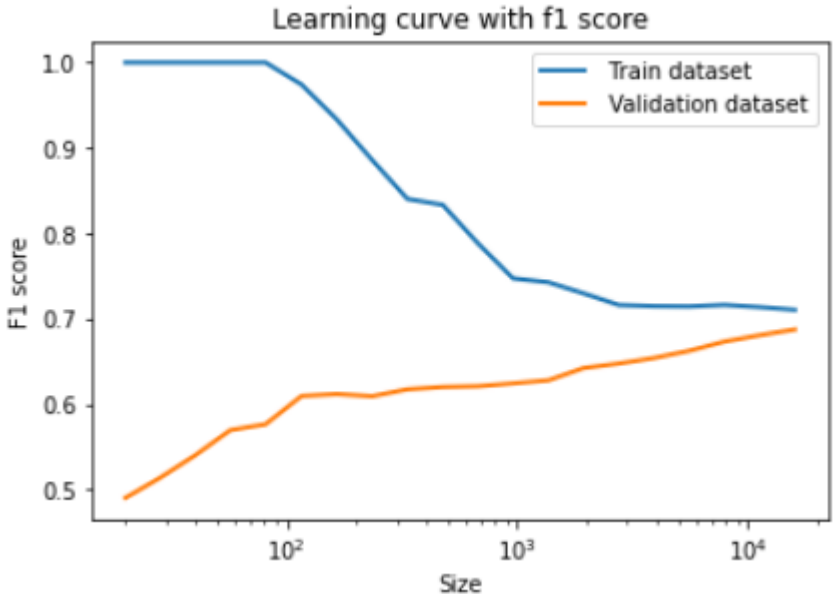


Παρατηρούμε ότι στα 2 datasets (train και validation) στο learning curve βλέπουμε πολύ πιο ομαλή κάθοδος του f1 score του train set και τα f1 scores χαμηλώνονται ελάχιστα με εκείνο χωρίς “balanced” ενώ η τελική απόκλιση παραμένει η ίδια.

Επίσης θα πειραματιστούμε για διάφορες τιμές του C το οποίο αλλάζει το βάρους του normalization.

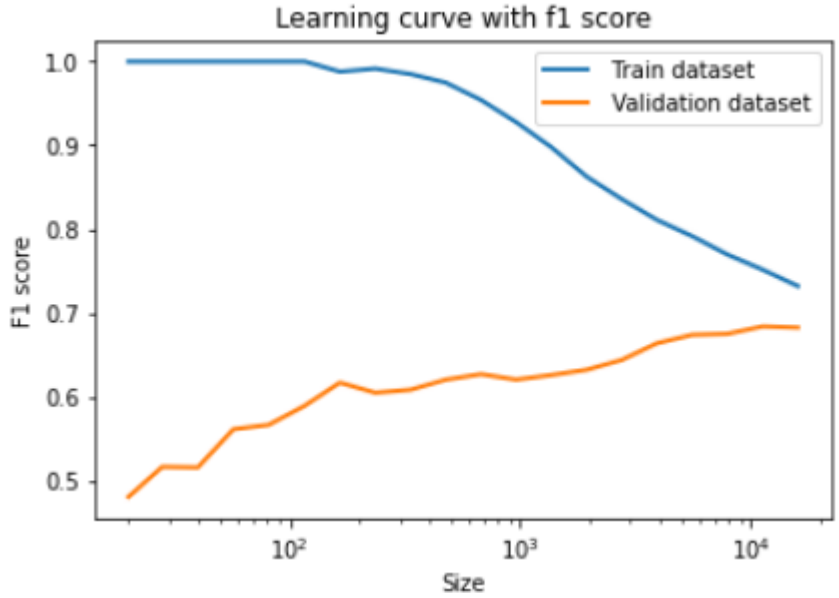
Για C = 0.2 έχουμε train f1 score 0.710 και validation f1 score 0.687 με learning curve:

Train	0.710
Validation	0.687



Για C = 3 έχουμε train f1 score 0.732 για validating f1 score 0.681 και learning curve:

Train	0.732
Validation	0.681



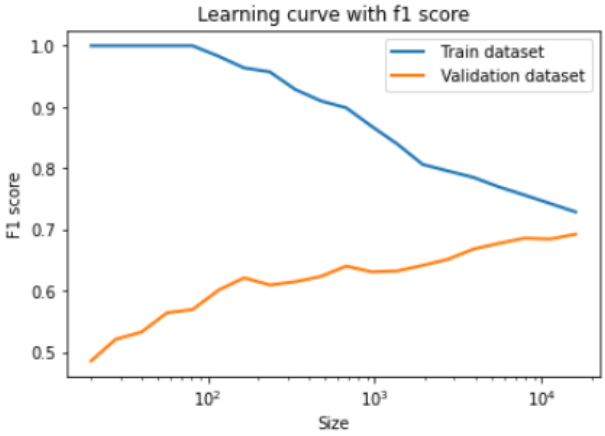
Παρατηρούμε ότι για μικρές τιμές του C τα f1 score παραμένουν σχεδόν τα ίδια με μια μικρή κάθοδος ενώ για μεγάλες τιμές αυξάνεται η απόκλιση μεταξύ τους ενώ ταυτόχρονα μειώνεται το score του validation, οπότε κρατάμε το default 1. Οι άλλες παράμετροι είναι είτε για διαφορετικό solver που δεν μπορούμε να χρησιμοποιήσουμε, είτε δεν αλλάζουν καθόλου τα δεδομένα οπότε και τα αγνοούμε.

Τελικό μοντέλο

Μέσω των πειραματισμών με διαφορετικές τεχνικές και παραμέτρους που έχουν αναφερθεί στην εργασία αυτή συμπεραίνουμε ότι σαν καλύτερο μοντέλο είναι pre-processing με lemmatization, tfidfvectorizer και LogisticRegression με τις παραμέτρους που είδαμε. Οπότε για το validation που δόθηκε έχουμε:

```
f1 score: 0.7290000324228499
precision score: 0.7558697682580355
recall score: 0.7186404606910366

VAL: f1 score: 0.692637376258634
VAL: precision score: 0.7155970744645627
VAL: recall score: 0.6822962313759859
```



Πρώτο υποερώτημα

Από θεωρία ξέρουμε ότι:

$$MSE(w) = \frac{1}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^2 \quad (i)$$

Επίσης $a^T a = [a_1\ a_2\ a_3\ \dots\ a_n]\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_n \end{bmatrix} = [a_1^2 + a_2^2 + a_3^2 + a_n^2]$

Από (i) έχουμε:

$$MSE(w) = \frac{1}{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^T\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)$$

$$= \frac{1}{m}\left(h_w\left(x^{(i)}\right)^T - y^{(i)T}\right)\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)$$

$$= \frac{1}{m}\left(h_w\left(x^{(i)}\right)^T h_w\left(x^{(i)}\right) - h_w\left(x^{(i)}\right)^T y^{(i)} - h_w\left(x^{(i)}\right)y^{(i)T} + y^{(i)T}y^{(i)}\right)$$

$$= \frac{1}{m}\left(h_w\left(x^{(i)}\right)^T h_w\left(x^{(i)}\right) - 2h_w\left(x^{(i)}\right)^T y^{(i)} h_w\left(x^{(i)}\right)y^{(i)T} + y^{(i)T}y^{(i)}\right)$$

Οπότε το gradient υπολογίζεται από:

$$\nabla_w MSE(w) = \frac{1}{m}(\nabla h_w\left(x^{(i)}\right)^T h_w\left(x^{(i)}\right) - 2\nabla h_w\left(x^{(i)}\right)^T y^{(i)} h_w\left(x^{(i)}\right) y^{(i)T} + \nabla y^{(i)T} y^{(i)})$$

$$= \frac{1}{m}(2X^T Xw - 2X^T y + 0)$$

Άρα $\nabla_w MSE(w) = \frac{2}{m}(X^T(Xw - y))$