# 

 Tweet Sentiment Analysis

Μηχανική Μάθηση

Εργασία Εξαμήνου

Τζανής Νικόλαος

ΑΜ: mtn2217

**Περιεχόμενα**

Εισαγωγή 03

Απαραίτητα Libraries και Dataset 04

Εξερεύνηση και Επεξεργασία των Δεδομένων 05

Δημιουργία του Feature Matrix και Oversampling 12

Helper Functions 14

Classification Models 15

SVM 15

KNN 17

Decision Trees 19

Random Forest 21

AdaBoost 23

Gaussian Naive Bayes 25

Κώδικας Demo και Περαιτέρω Testing 27

**Εισαγωγή**

Σήμερα, τα μέσα κοινωνική δικτύωσης έχουν εξελιχθεί σε αναπόσπαστο κομμάτι της καθημερινότητας. Ειδικά το Twitter, αποτελεί πλέον μία πλατφόρμα όπου οι περισσότεροι άνθρωποι, καθώς και αρκετές εταιρείες, οργανώσεις ή ακόμα και κρατικοί φορείς μπορεί να εκφράσουν τις απόψεις, τις σκέψεις και τα συναισθήματά τους για διάφορα ζητήματα. Η ανάλυση των συναισθημάτων στα παραπάνω tweets αποτελεί πλέον ένα σημαντικό εργαλείο για διάφορες εταιρείες καθώς τους επιτρέπει να κατανοήσουν καλύτερα τις απόψεις των πελατών τους, αλλά και τα συναισθήματά τους για τα προϊόντα και τις υπηρεσίες που προσφέρουν.

Οι αλγόριθμοι μηχανικής μάθησης μάς έχουν επιτρέψει να αναλύσουμε και να κατηγοριοποιήσουμε τα tweets ανάλογα με το συναίσθημα που εκφέρουν χρησιμοποιώντας διάφορες πρακτικές επεξεργασίας φυσικής γλώσσας (Natural Language Processing – NLP). Αναλύοντας τα αποτελέσματα των παραπάνω αναλύσεων, οι εταιρείες μπορούν να κάνουν κατάλληλες τροποποιήσεις στα προϊόντα και τις υπηρεσίες τους, έτσι ώστε να είναι περισσότερο φιλικά προς τους χρήστες.

Η πρόκληση στην ανάλυση του συναισθήματος των tweet έγκειται στην ακριβή ανίχνευση του τόνου ενός tweet και των συμφραζομένων, ιδιαίτερα όταν υπάρχει σαρκασμός ή ειρωνεία. Παρά τις παραπάνω προκλήσεις οι αλγόριθμοι μηχανικής μάθησης συχνά καταφέρουν να ταξινομήσουν με μεγάλη ακρίβεια τα συναισθήματα των tweets.

Η παρούσα ανάλυση θα εξετάσει διάφορες τεχνικές και μοντέλα για την ανάλυση των συναισθημάτων στα tweets και την ικανότητά των μοντέλων αυτών να εξάγουν ακριβή αποτελέσματα.

**Απαραίτητα Libraries και Dataset**

**Libraries**

Ορισμένα libraries που χρησιμοποιούνται ευρέως στην παρούσα ανάλυση αναφέρονται παρακάτω, καθώς και στο αρχείο Requirements.txt, το οποίο βρίσκεται στο github repository της ανάλυσης.

* **Pandas:** Ένα library για την ανάλυση και τροποποίηση δεδομένων. Έχει δομές δεδομένων για την αποθήκευση μεγάλου όγκου πληροφορίας και αρκετά functions για την τροποποίηση των δεδομένων.
* **Numpy:** Ένα library για αριθμητικούς υπολογισμούς. Περιέχει αρκετά functions για μαθηματικές πράξεις.
* **Sklearn** ή Scikit-learn: Ένα library που περιέχει διάφορα functions για τεχνικές μηχανικής μάθησης, καθώς και πιο ειδικές βιβλιοθήκες για κάθε μοντέλο.
* **Matplotlib:** Ένα library για οπτικοποίηση δεδομένων. Περιέχει πολλά functions για τη δημιουργία διαγραμμάτων.
* **Seaborn:** Ακόμα ένα library για οπτικοποίηση δεδομένων που έχει βασιστεί στο matplotlib.

**Dataset**

Το dataset που χρησιμοποιήθηκε στην ανάλυση προέρχεται από την ιστοσελίδα Kaggle και βρίσκεται στο φάκελο “datasets” του repository.

Στο jupyter notebook που χρησιμοποιούμε το dataset «διαβάζεται» με την παρακάτω εντολή και αρχικά το ονομάζουμε “raw\_df”:



**Εξερεύνηση και Επεξεργασία των Δεδομένων**

**Διαστάσεις του Dataframe**

Τα δεδομένα από το dataset που χρησιμοποιούμε έχουν εξαχθεί σε μορφή dataframe και είναι:



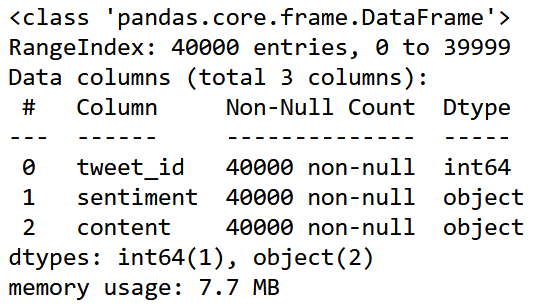
που σημαίνει ότι έχουμε συνολικά 40.000 tweet και 3 στήλες.

**Επισκόπηση/Γενικά Στοιχεία για το Dataframe**

Με το function “info” της Pandas παίρνουμε κάποια ενδιαφέροντα δεδομένα για το Dataframe που χρησιμοποιούμε:



Το αποτέλεσμα της παραπάνω εντολής:



Μας λέει πρακτικά ότι έχουμε 3 στήλες στο dataframe, οι οποίες έχουν τίτλους ‘tweet-id’, ‘sentiment’ και ‘content’, αντίστοιχα.

Μας πληροφορεί ακόμη ότι ο τύπος των δεδομένων της κάθε στήλης είναι για τις 3 παραπάνω στήλες ‘int64’, ‘object’ και ‘object’.

Σημείωση: Στην pandas τα strings λογίζονται ως objects.

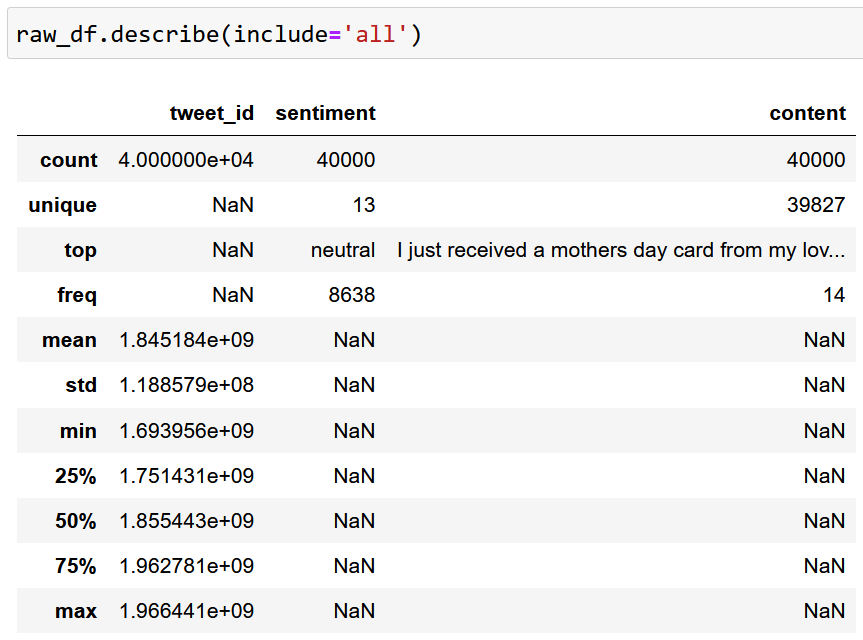
Φαίνεται επίσης ποιες τιμές δεν είναι null (θα χρειαστούμε αυτή την πληροφορία αργότερα).

Τέλος αναφέρεται ο όγκος μνήμης που χρησιμοποιεί το dataset.

**Το Function “Describe”**

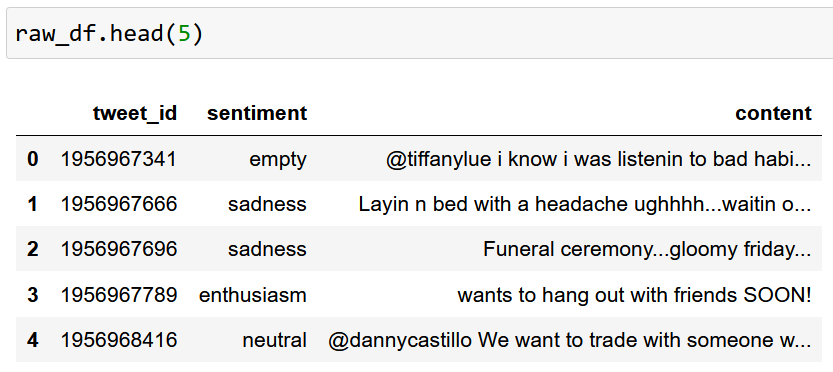
Με το function αυτό συνήθως μπορούμε να εξάγουμε ορισμένα ενδιαφέροντα δεδομένα για το dataset που έχουμε, ωστόσο είναι περισσότερο χρήσιμο για datasets με int ή float και όχι τόσο για datasets με strings, όπως το δικό μας.

Στην περίπτωσή μας είναι ίσως χρήσιμο για να αποτυπωθούν οι 13 κατηγορίες συναισθημάτων που θα εξετάσουμε.



**Το Function “Head”**

Το function αυτό της Pandas μάς επιστρέφει τις 5 πρώτες γραμμές του dataset έτσι ώστε να έχουμε μία καλύτερη εικόνα των δεδομένων κάθε στήλης.

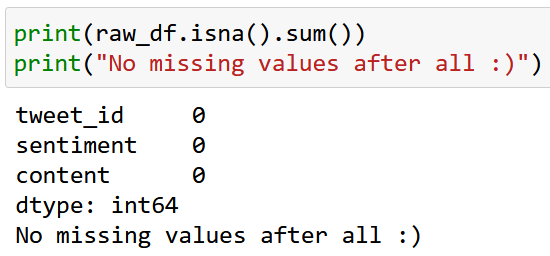


**Ελλιπή Δεδομένα**

Για να αντιμετωπίσουμε την έλλειψη δεδομένων, όταν για παράδειγμα δεν έχουμε sentiment σε ένα tweet ή όταν δεν υπάρχει value στη στήλη content ενώ έχουμε τιμή για το sentiment, έχουμε δύο επιλογές:

* Να αφαιρέσουμε τη γραμμή στην οποία λείπει ένα από τα παραπάνω δύο values (το sentiment ή το content)
* Να προσπαθήσουμε να «μαντέψουμε» ποια θα ήταν τα δεδομένα που λείπουν (data imputation)

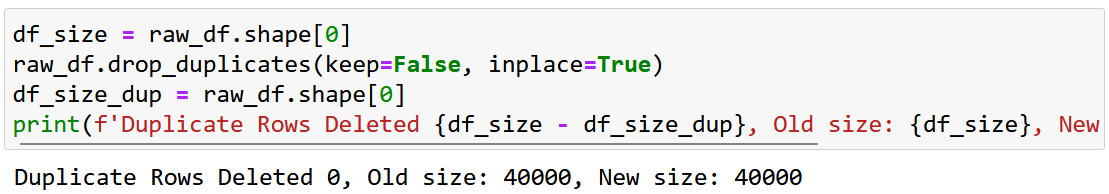
Αρχικά με το function isna της pandas εξετάζουμε αν έχουμε πράγματι ελλιπή δεδομένα. Το function αυτό επιστρέφει ένα πίνακα με true/false ανάλογα αν έχουμε value σε αυτό το column, οπότε εδώ προσθέτουμε και ένα sum στο τέλος για να δούμε πόσα λείπουν σε κάθε column.



Εδώ ευτυχώς δεν είχαμε missing values.

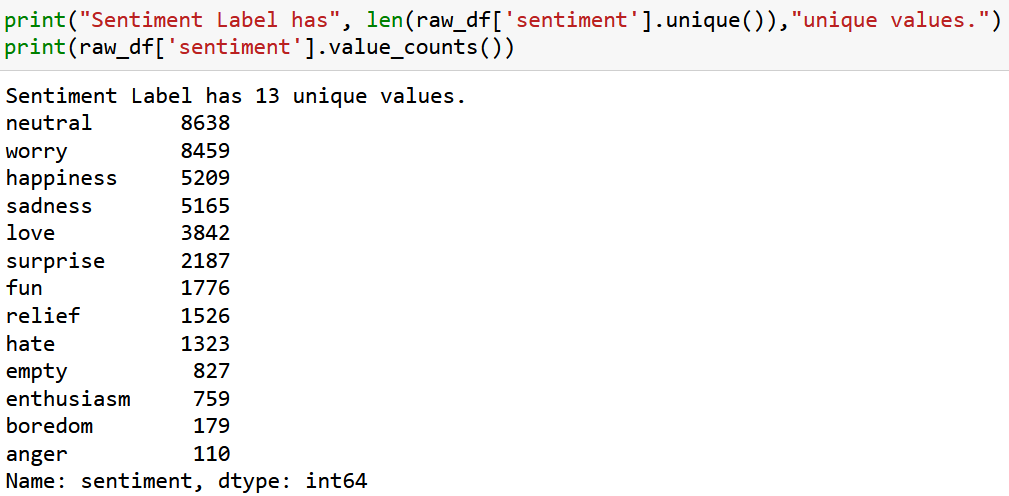
**Αφαίρεση Διπλοτύπων Γραμμών**

Σε ορισμένες περιπτώσεις μπορεί να επαναλαμβάνονται γραμμές σε ένα dataset (δηλαδή να έχουμε ακριβώς την ίδια γραμμή 2 ή περισσότερες φορές). Σε αυτή την περίπτωση χρησιμοποιούμε το function “drop\_duplicates” της pandas (με το inplace=True αλλάζουμε απευθείας το dataset αφαιρώντας τις διπλότυπες γραμμές). Εδώ δεν είχαμε καθόλου επαναλήψεις γραμμών.



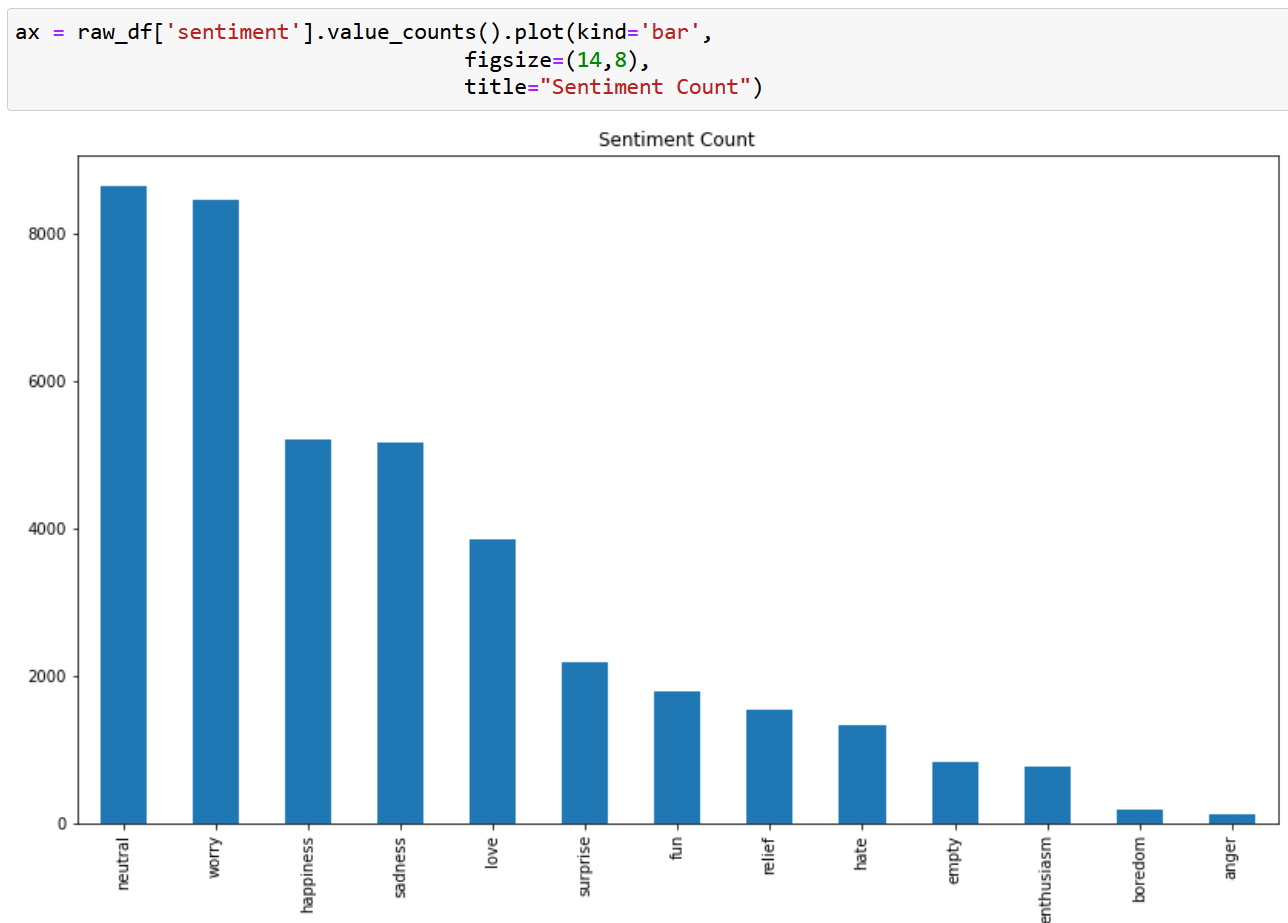
**Εξέταση του Αριθμού των Tweets Κάθε Κλάσης**

Στην περίπτωσή μας οι κλάσεις που εξετάζουμε είναι τα διαφορετικά sentiments, επομένως θα έχουμε 13 διαφορετικές κλάσεις. Με το function “unique” της pandas βλέπουμε πόσα είναι τα διαφορετικά values στη στήλη ‘sentiment’ και ακολούθως με την ‘value\_counts’ βλέπουμε πόσα tweets αντιστοιχούν στο κάθε sentiment συνολικά.



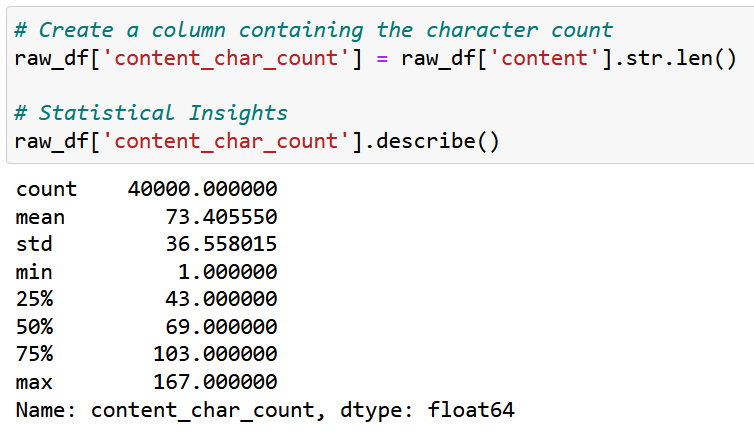
Ιδανικά, θέλουμε οι κλάσεις που εξετάζουμε να έχουν παρόμοιο αριθμό tweets, έτσι ώστε να γίνεται καλύτερο classification και να μην υπάρχει bias. Θα αντιμετωπίσουμε αργότερα το παραπάνω πρόβλημα.

Από την οπτικοποίηση των αποτελεσμάτων σε ένα bar chart, φαίνεται εμφανώς ότι ορισμένες κλάσεις έχουν πολύ περισσότερα tweets σε σχέση με άλλες:



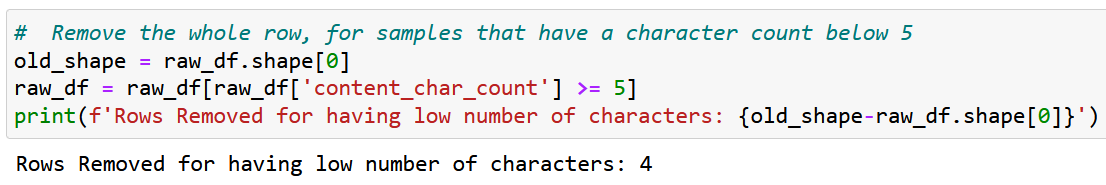
**Αφαίρεση Outliers από το Dataset**

Θεωρούμε ότι tweets με αριθμό χαρακτήρων μικρότερο του 5 δεν αποτελούν πραγματικά tweets και επομένως τα αφαιρούμε. Για να επιτύχουμε τον παραπάνω στόχο αρχικά δημιουργούμε μία νέα στήλη που περιέχει τον αριθμό των χαρακτήρων σε κάθε tweet (από τη στήλη ‘content’):

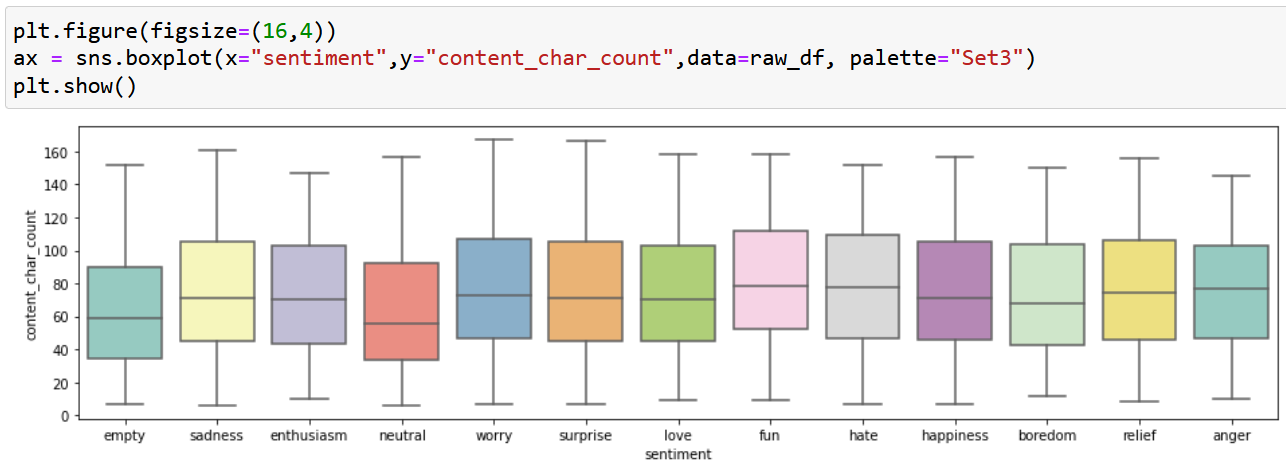


Και με το ‘describe’ εξάγουμε κάποια ενδιαφέροντα δεδομένα, όπως για παράδειγμα το μέσο μήκος των tweets.

Στη συνέχεια, αν κάποιο tweet έχει λιγότερους από 5 χαρακτήρες, το αφαιρούμε:

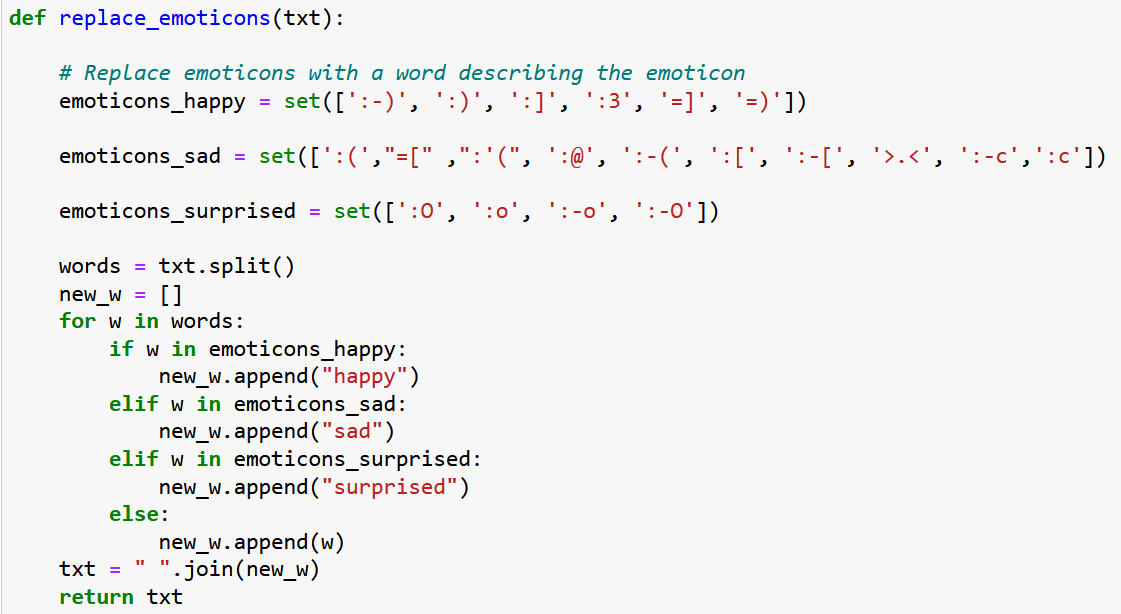


Με το seaborn μπορούμε να κάνουμε μία καλή οπτικοποίηση των δεδομένων σχετικά με το μήκος των tweets:



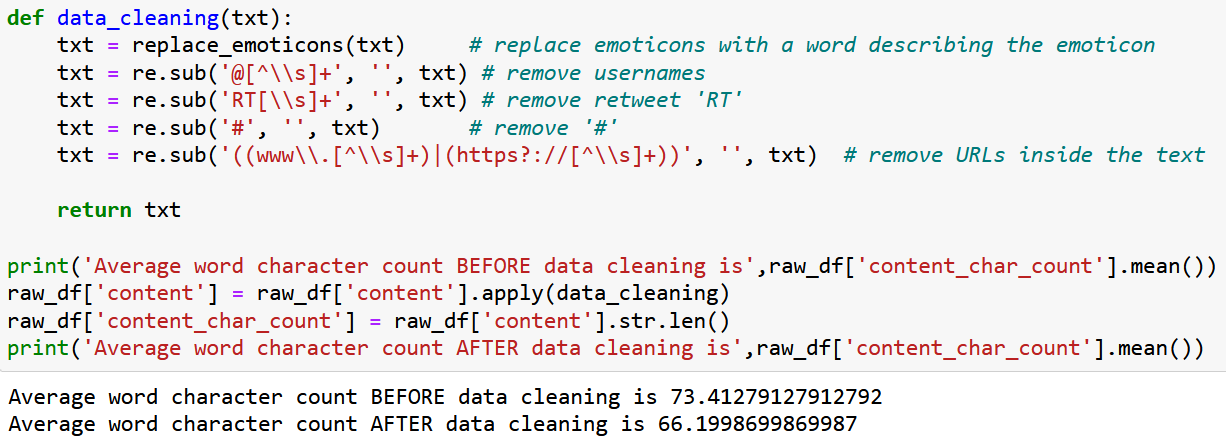
**Άντληση Πληροφορίας από Emojis**

Καθώς τα emojis θεωρούνται ως επί το πλείστον σημεία στίξης, έκανα μία προσπάθεια να αντλήσω πληροφορία και από αυτά σε κάποιο βαθμό. Με το παρακάτω function που δημιούργησα αντικατέστησα τα συχνότερα emojis με το sentiment που αντιστοιχούν:



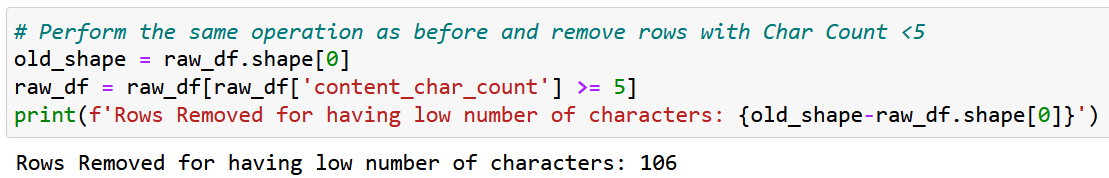
**Αφαίρεση Username/Retweet/URL**

Εφόσον τα username, τα retweets, τα url και το σύμβολο που προηγείται των hashtags (#) δεν προσφέρουν κάποια ουσιώδη πληροφορία για το classification, τα αφαίρεσα εντελώς. Χρησιμοποιώντας regular expressions στη συνάρτηση data\_cleaning, έκανα όλα τα παραπάνω, εφαρμόζοντας επίσης τη συνάρτηση που προαναφέρθηκε για τη μετατροπή των emoji σε λέξεις:



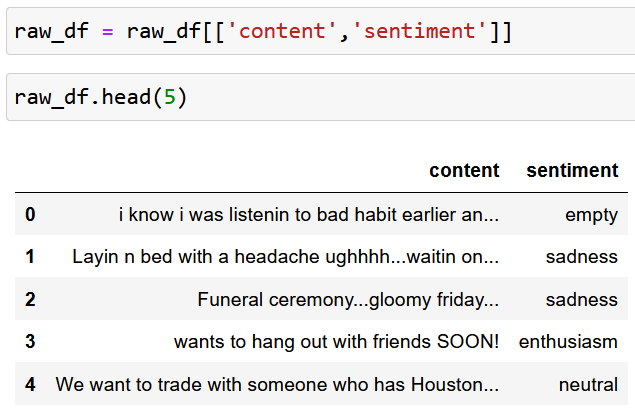
Προφανώς, με την εφαρμογή της παραπάνω συνάρτησης βλέπουμε ότι το μέσο μέγεθος του tweet μειώθηκε.

Στη συνέχεια, ελέγχουμε και πάλι αν έχουμε γραμμές με λιγότερους από 5 χαρακτήρες ανά tweet και τις αφαιρούμε:



**Αφαίρεση Περιττών Στηλών**

Οι μοναδικές στήλες που χρειαζόμαστε πλέον είναι οι ‘content’ και ‘sentiment’ οπότε δημιουργούμε ένα νέο dataframe που περιέχει μόνο αυτές τις στήλες:



**Label Encoding**

Στο σημείο αυτό είναι σημαντικό να αναφερθεί ότι στο training των μοντέλων δεν μπορούμε να χρησιμοποιήσουμε strings για το y και είναι απαραίτητο να τα μετατρέψουμε σε άλλη μορφή.

Μία απλή στρατηγική θα ήταν να αναθέσουμε στις τιμές του y, δηλαδή στα sentiments τιμές από το 0 έως το 12. Αυτό γίνεται με το Label Encoding, τελικά ωστόσο δεν το έχω κάνει γιατί γινόταν αυτόματα από τα διάφορα models.

Άλλες προσεγγίσεις είναι το Ordinal Encoding, το οποίο χρησιμοποιείται όταν έχει σημασία ο αριθμός που μπαίνει στο y, όπως για παράδειγμα σε ratings όπου πχ το 0/5 είναι λογικό να αντιστοιχηθεί στο 0, το 1 στο 1 κλπ

Τέλος, υπάρχει και το OneHot Encoding το οποίο λειτουργεί με μοναδιαίους πίνακες.

**Δημιουργία του Feature Matrix και Oversampling**

**Feature Matrix**

Η προσέγγιση που χρησιμοποιώ για τη δημιουργία του feature matrix είναι το Bag of Words (BoW). Πρακτικά, το TfidVectorizer της sklearn δημιουργεί ένα μεγάλο feature matrix. Στον πίνακα αυτό πρακτικά έχουμε κάθε tweet στις γραμμές και βάζουμε ‘1’ αν περιέχει την κάθε λέξη στις στήλες.

Η TfidfVectorizer εκτελεί επίσης και άλλες λειτουργίες μέσω των parameters στην κλήση της, όπως για παράδειγμα να μετατρέψει όλα τα γράμματα από κεφαλαία σε μικρά (‘lowercase’: True) και να αφαιρέσει τα σημεία στίξης (‘stopwords’).

Με το ‘ngram\_range’ επιλέγουμε αν θα υπάρχουν και δύο συνεχόμενες λέξεις στα features (εδώ έχουμε επιλέξει ναι), καθώς και το μέγιστο αριθμό των features (έχουμε επιλέξει 1200).

Είναι σημαντικό να σημειωθεί ότι το TfidfVectorizer εφαρμόζει και το νόμο του Zipf, δηλαδή δίνει λιγότερη αξία στις λέξεις που επαναλαμβάνονται πολύ και μεγαλύτερη σε λέξεις με λίγες εμφανίσεις.

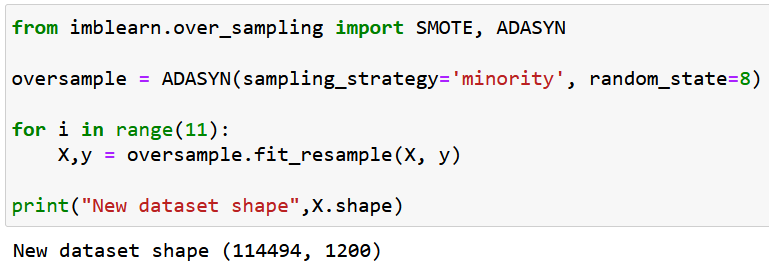


**Oversampling**

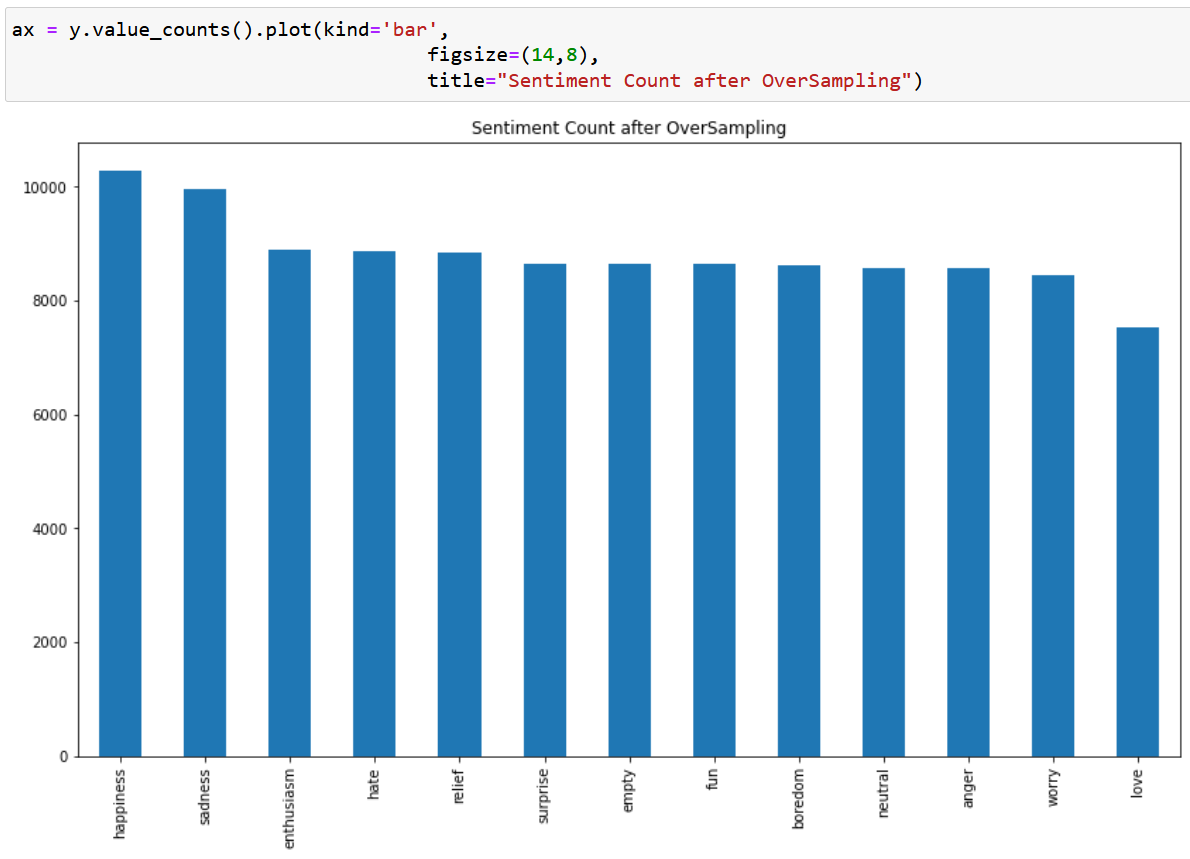
Στο σημείο αυτό γίνεται «διόρθωση» στον αριθμό των entries κάθε κλάσης, προκειμένου να αντιμετωπιστεί η μεγάλη διάφορα ανάμεσα στις κλάσεις που εξετάζουμε. Μία από τις τεχνικές που μπορεί να χρησιμοποιηθεί είναι η SMOTE, η οποία δημιουργεί περισσότερα entries για την κλάση που μειοψηφεί. Στη συγκεκριμένη ανάλυση έχω χρησιμοποιήσει την ADASYN, που είναι μία πιο γενική μορφή του αλγόριθμου της SMOTE η οποία προσαρμόζει επίσης κατάλληλα τον αριθμό των entries που δημιουργούνται ανάλογα με τη «δυσκολία» δημιουργίας νέων entry σε κάθε κλάση.

Και οι δύο παραπάνω αλγόριθμοι έχουν ως στόχο να εξισορροπήσουν τη διαφορά των entries ανάμεσα σε δύο κλάσεις με μεγάλες διαφορές.

Αναφορικά με τα params της ADASYN, έχω χρησιμοποιήσει το sampling\_strategy ‘minority’ 11 φορές γιατί τόσο το ‘majority’ όσο και το ‘auto’ μου έβγαζαν προβλήματα.



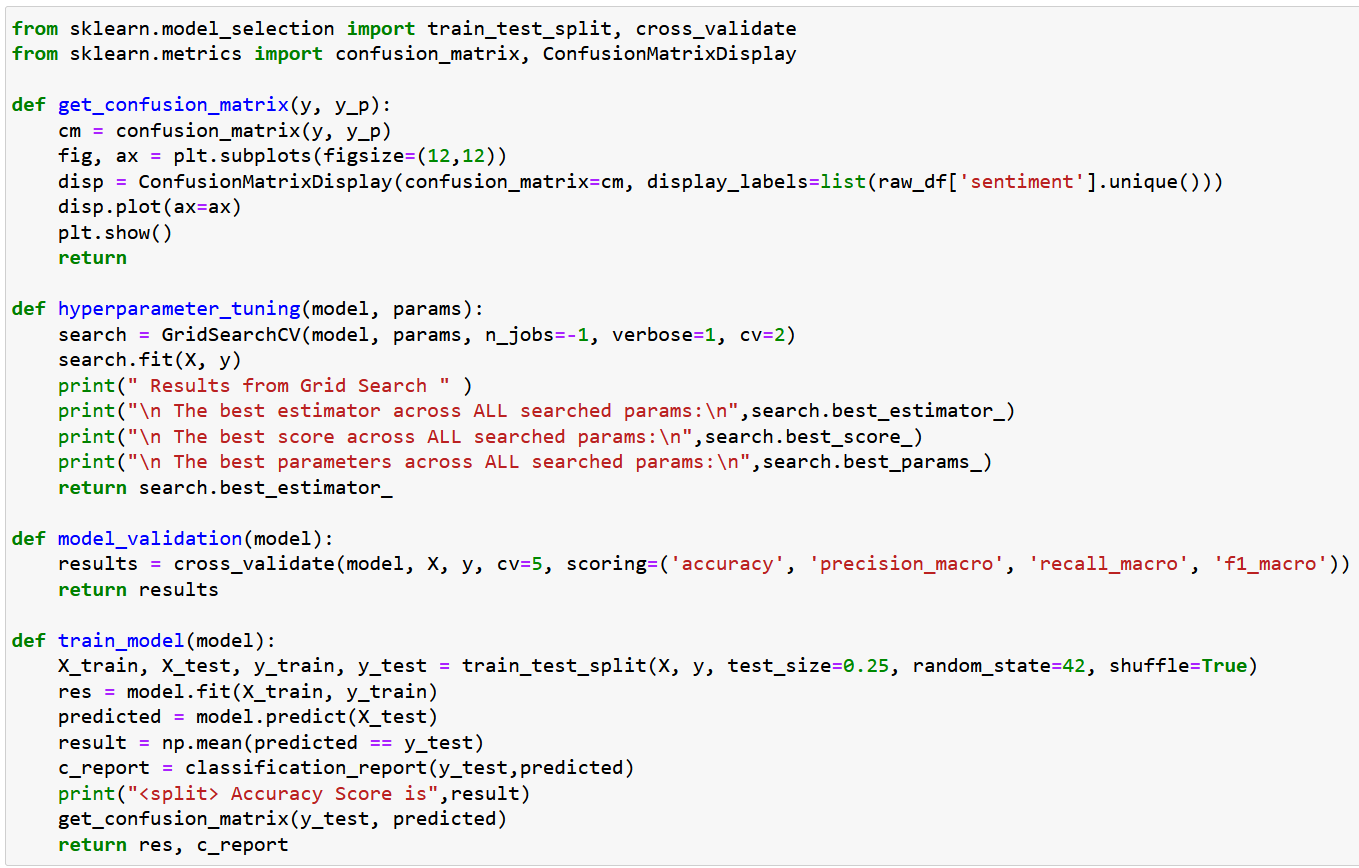
Δημιουργώντας ένα νέο bar graph βλέπουμε ότι πλέον το count της κάθε κλάσης είναι συγκρίσιμο με τις υπόλοιπες:



**Helper Functions**

Πριν ξεκινήσω την εφαρμογή των διαφόρων μοντέλων για την ανάλυση των δεδομένων, δημιουργώ 4 helper functions που θα βοηθήσουν σε αυτή τη διαδικασία. Τα functions αυτά είναι:

1. **get\_confusion\_matrix**. Το function αυτό επιστρέφει το confusion matrix από το οποίο φαίνονται τα σωστά και τα λανθασμένα predictions του classifier που χρησιμοποιήθηκε.
2. **hyperparameter\_tuning**. Το function αυτό ελέγχει διάφορα parameters που έχουμε δώσει για το μοντέλο που θέλουμε να εξετάσουμε. Στη συνέχεια χρησιμοποιεί την κλάση GridSeachCV της scikit-learn για να μας επιστρέψει τις καλύτερες τιμές για τα params από αυτές που έχουμε δώσει.
3. **model\_validation**. Cross validation του μοντέλου και εξαγωγή δεδομένων για accuracy, precision, recall και macro-f1. Η τιμή 5 του cv ουσιαστικά δείχνει ότι γίνεται cross validation με 5 folds (4 training – 1 test).
4. **train\_model**. Χωρίζει τα δεδομένα σε train και test (3:1) και κάνει το τελικό training του μοντέλου με βάση τα ιδανικά params που υπολογίσαμε στις προηγούμενες συναρτήσεις.



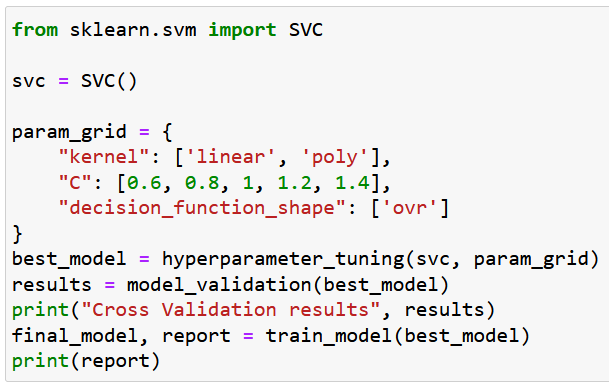
**Classification Models**

**SVM**

Το πρώτο μοντέλο που θα εξετάσω είναι το SVM. Το μοντέλο αυτό πρακτικά προσπαθεί να διαχωρίσει τις διάφορες κλάσεις με μία γραμμή (ή επίπεδο). Η παραπάνω γραμμή μεγιστοποιεί τις αποστάσεις στα κοντινότερα σημεία μεταξύ διαφορετικών κλάσεων και μπορεί έτσι να εκτιμήσει σε ποια κλάση θα ανήκει ένα νέο δεδομένο που θα δώσουμε.

Η σημαντικότερη παράμετρος σε αυτό το μοντέλο είναι το ‘C’, το οποίο πρακτικά αντικατοπτρίζει την «ανοχή» του μοντέλου σε misclassifications στο training set. Συχνά, όταν επιτρέπουμε στο C να έχει χαμηλές τιμές (δηλαδή να επιτρέπει misclassifications στο training set) επιτυγχάνουμε καλύτερο generalization. Στο συγκεκριμένο παράδειγμα εξετάζουμε τις τιμές του C 0.6, 0.8, 1, 1.2 και 1.4.

Μία άλλη παράμετρος που εξετάζουμε στο hyperparameter tuning είναι το kernel που παίρνει τιμές ‘linear’ ή ‘poly’. Αυτό σημαίνει ότι εξετάζουμε αν μπορούμε να διαχωρίσουμε γραμμικά ή πολυωνυμικά (υψώνουμε τα δεδομένα σε κάποια δύναμη).

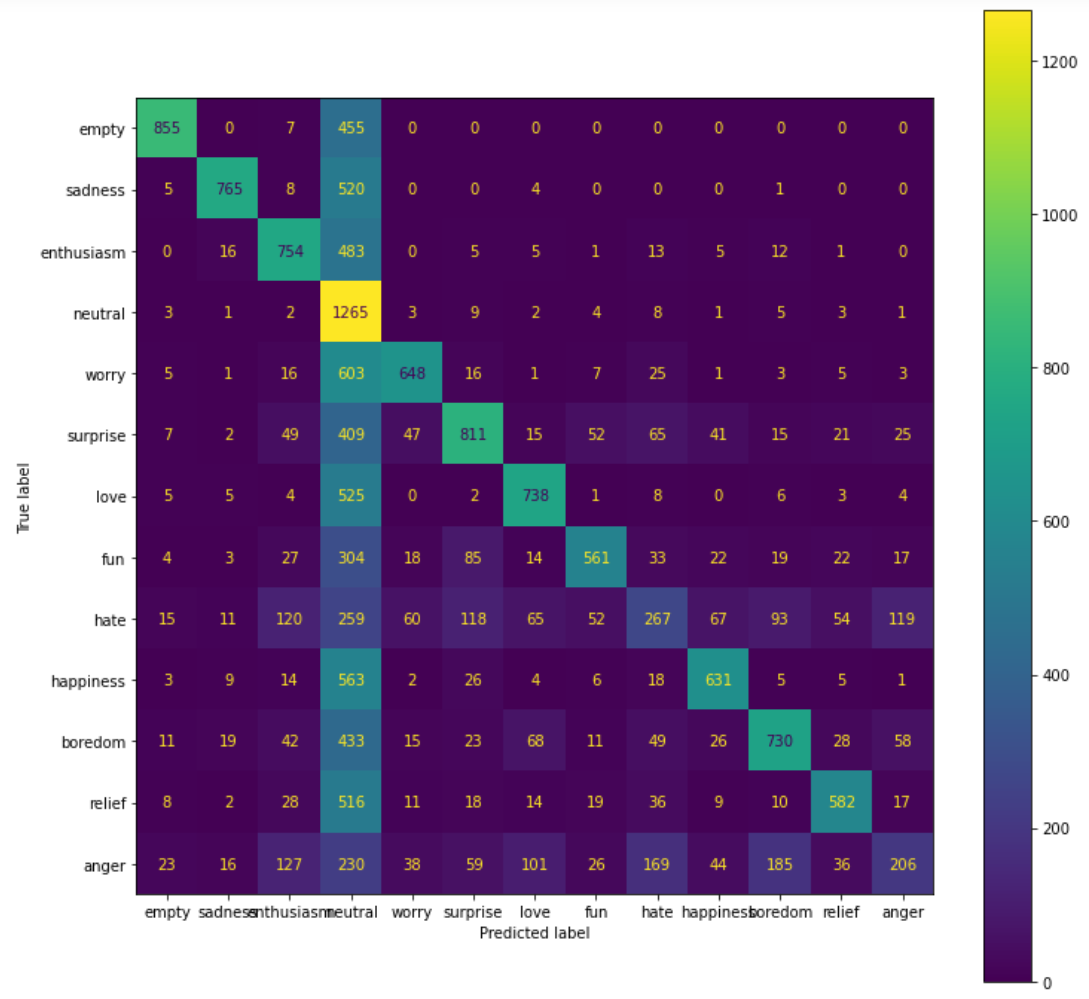


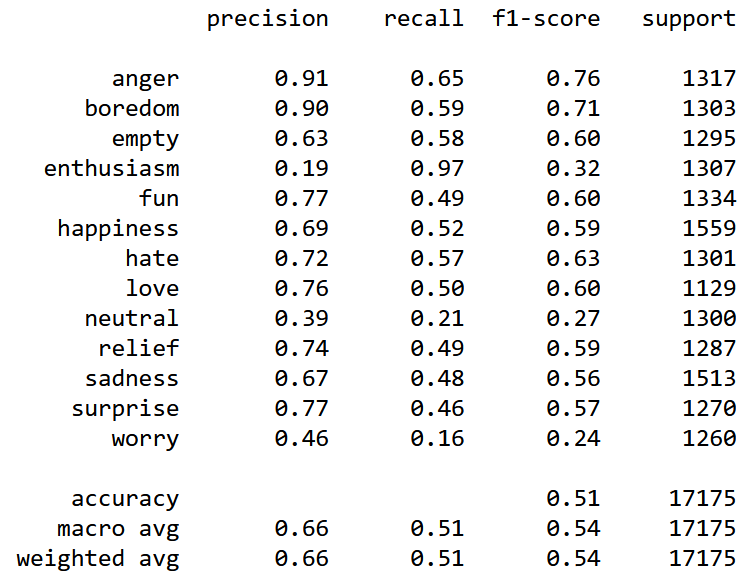
Τα καλύτερα params τελικά σύμφωνα με το tuning είναι τα:



Με αυτά τα parameters επιτυγχάνουμε accuracy περίπου 67% στο training και 51% στο validation.

Παραθέτω επίσης το confusion matrix και τους πίνακες με τα precision, recall και f1-score κάθε κλάσης:





**kNN**

Το επόμενο μοντέλο που θα εξετάσω είναι το kNN (k-nearest neighbors) . Το μοντέλο αυτό εξετάζει σε ποια κλάση ανήκει το tweet ανάλογα με τα «γειτονικά» του data points. Για παράδειγμα αν εξετάζουμε 5 γειτονικά data points και έχουμε 3 ‘angry’ και 2 ‘happy’ το tweet θα ταξινομηθεί ως ‘angry’.

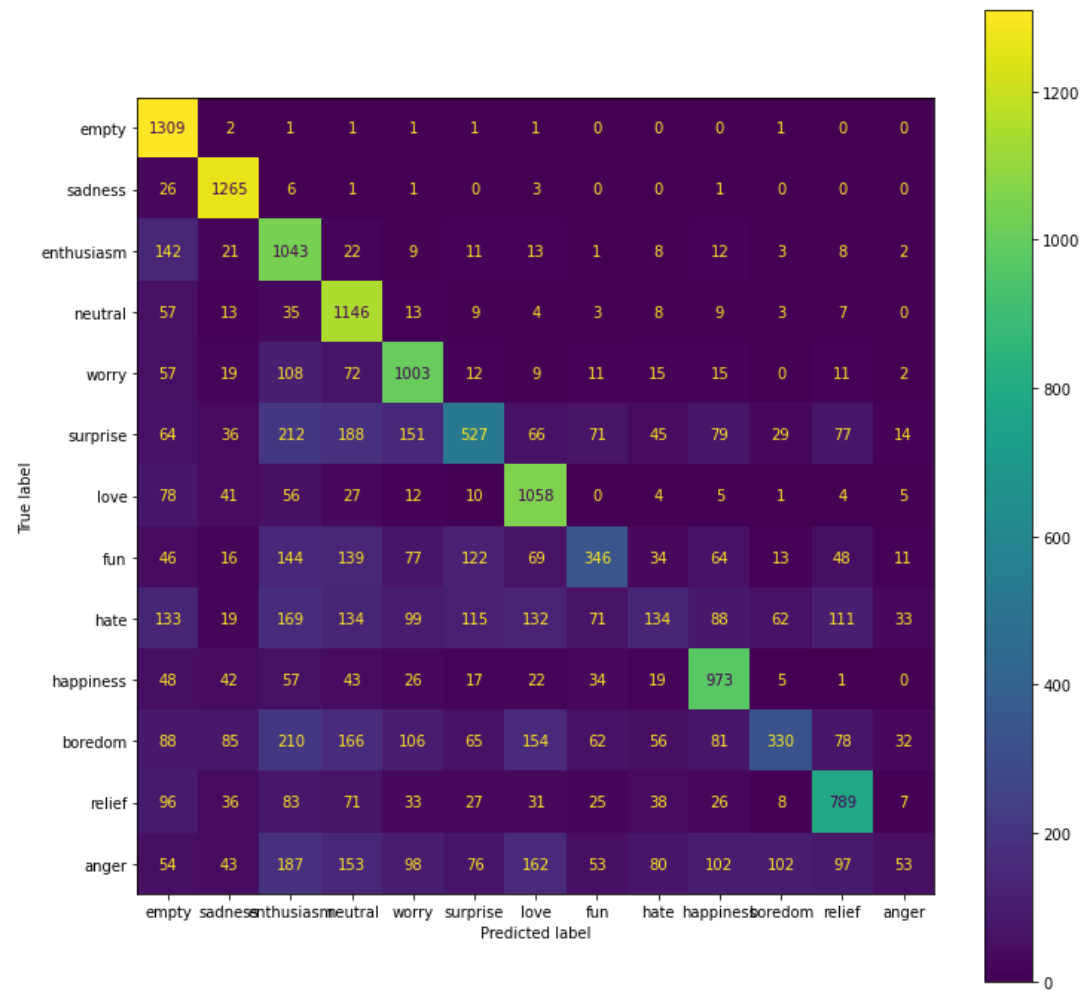
Εδώ προφανώς έχει σημασία πόσα είναι τα γειτονικά data points που θα εξετάσουμε καθώς μία μικρότερη τιμή του ‘n\_neighbors’ μπορεί να μας επιτρέψει να αντιληφθούμε πιο σύνθετες συσχετίσεις στα δεδομένα, ενώ μία μεγαλύτερη τιμή μπορεί να δημιουργήσει μεν ένα ισχυρότερο μοντέλο που δεν κάνει overfit, αλλά δεν αντιλαμβάνεται λεπτομέρειες στη σύνδεση των δεδομένων.

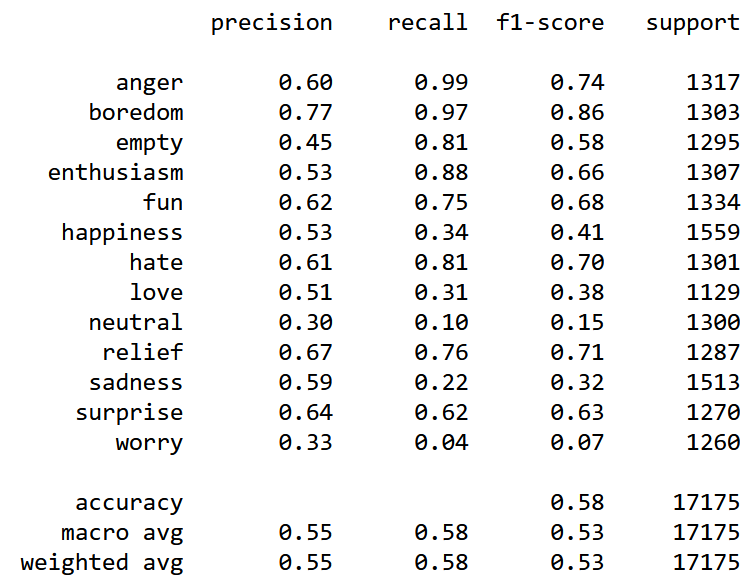
Εδώ δοκίμασα τιμές για το ‘n\_neighbors’ 3 και 5 και το hyperparameter tuning επέστρεψε ότι η καλύτερη από αυτές είναι το 3.



Με αυτά τα parameters επιτυγχάνουμε accuracy περίπου 74% στο training και 58% στο validation.

Παραθέτω επίσης το confusion matrix και τους πίνακες με τα precision, recall και f1-score κάθε κλάσης:





**Decision Trees**

Το επόμενο μοντέλο που θα εξετάσω είναι τα Decision Trees ή Δέντρα Απόφασης. Εδώ το σημαντικότερο param είναι το ‘max depth’. Γενικά όσο μεγαλύτερο είναι το max depth τόσο πιο πολύ ανεβαίνει το accuracy του μοντέλου για το dataset, ωστόσο υπάρχει κίνδυνος για overfit.

Άλλες παράμετροι είναι οι ‘min samples split’ που σημαίνει μέχρι ποιο αριθμό θα συνεχίσει να κάνει split τα κλαδιά του δέντρου (το default είναι 1). Όσο μικρότερο είναι το value αυτής της παραμέτρου τόσο αυξάνεται ο κίνδυνος overfit.

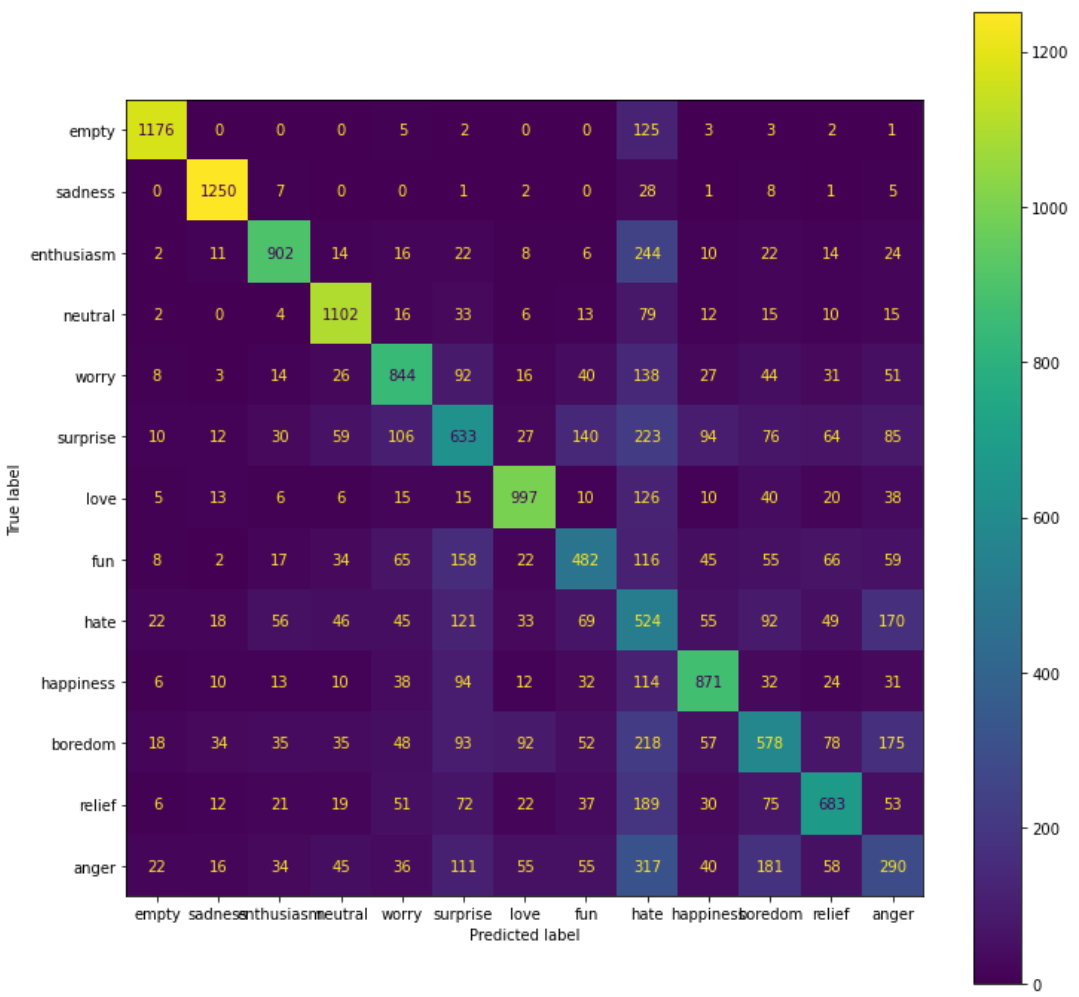
Μία άλλη παράμετρος είναι η ‘min samples leaf’. Οι μικρές τιμές σε αυτό το metric αυξάνουν την πιθανότητα overfit, ενώ σε μεγάλες τιμές υπάρχει πιθανότητα underfit.

Εδώ δοκίμασα τιμές για το max\_depth 100 και 500, για το min\_samples\_split 2 και 4 και για το min\_samples\_leaf 1, 2 και 4.

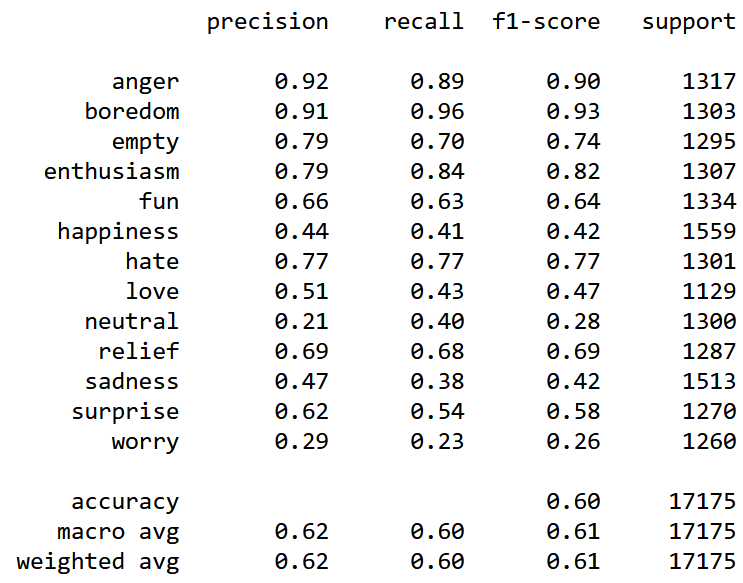


Με τα παραπάνω params πετυχαίνει accuracy 91% στο training και 60% στο validation.

Το confusion matrix ήταν:



Και τα υπόλοιπα δεδομένα:



**Random Forest**

Το επόμενο μοντέλο είναι το Random Forest. Το μοντέλο αυτό δημιουργεί πολλά μικρά decision trees και ακολούθως επιλέγει τυχαία μερικά από αυτά και βάσει αυτών δίνει ένα prediction για την κλάση.

Ο αλγόριθμος λέγεται “Random” Forest γιατί υπάρχει τυχαιότητα τόσο στην επιλογή των features που χρησιμοποιούνται για τη δημιουργία των decision trees όσο και στο κομμάτι του dataset που βασίζεται το decision tree.

Τα 2 σημαντικότερα params για αυτό το μοντέλο είναι το ‘n\_estimators’ που βασικά αναφέρεται στον αριθμό των δέντρων που χρησιμοποιούμε. Γενικά, όσο μεγαλύτερος είναι αυτός ο αριθμός τόσο πιο ακριβές θα είναι το μοντέλο.

Ένα άλλο σημαντικό metric είναι το ‘max\_depth’, το οποίο αναφέρεται στο μέγιστο βάθος που μπορεί να φτάσει το κάθε δέντρο. Όπως ισχύει και με τα decision trees, όσο πιο μεγάλο είναι το max\_depth τόσο περισσότερο αυξάνεται ο κίνδυνος overfit.

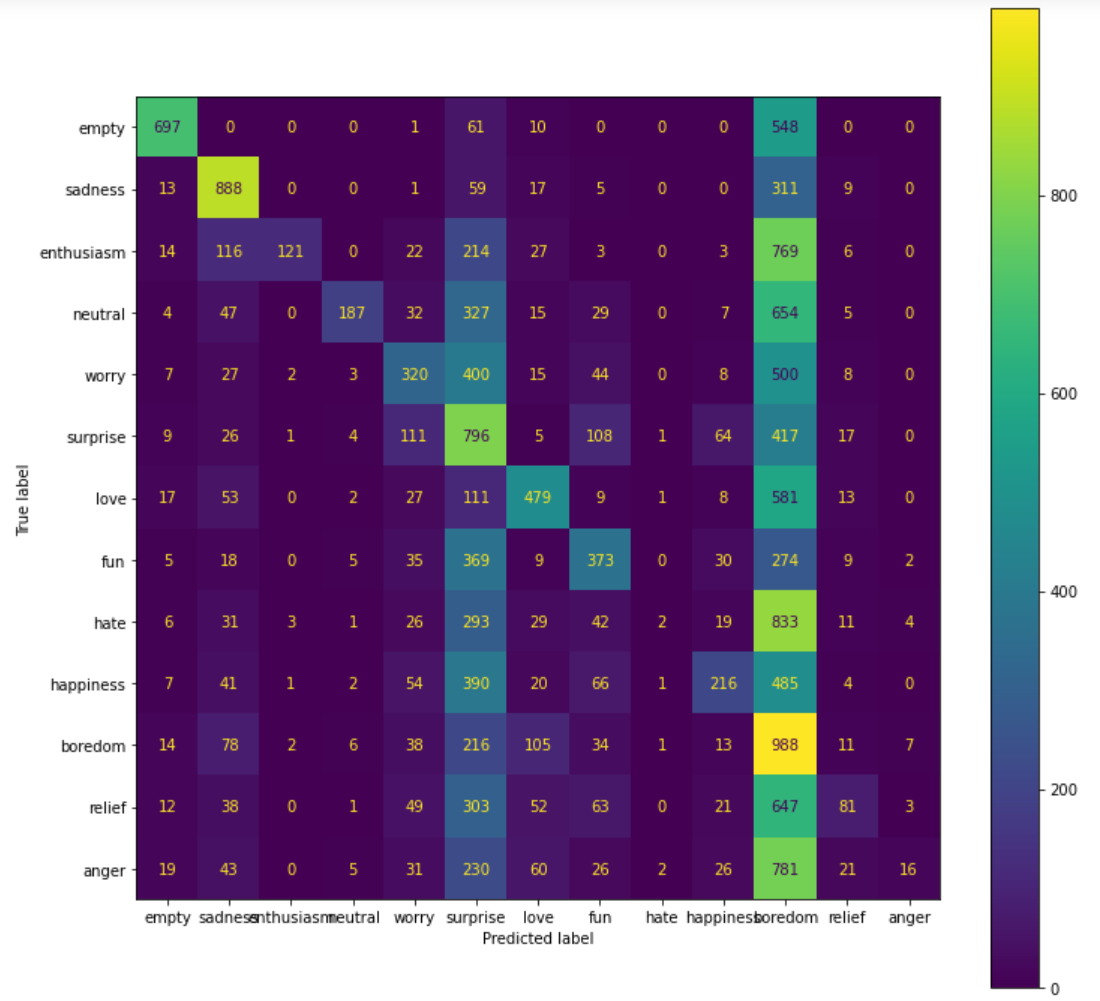
Δοκίμασα τιμές 100 και 200 για το n\_estimators και 50, 100 για το max\_depth.

Το αποτέλεσμα του hyperparameter tuning ήταν:

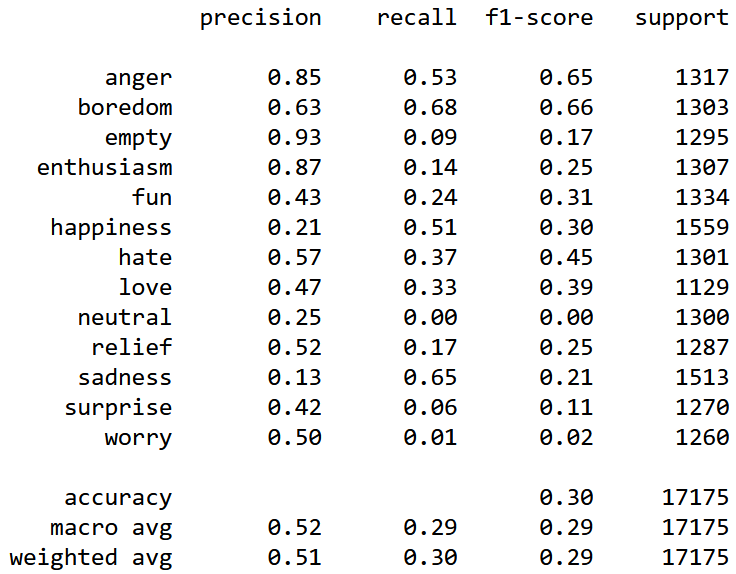


Το accuracy που πετυχαίνουμε με τα παραπάνω params είναι περίπου 31% για το training και 30% για το validation.

Το confusion matrix είναι το εξής:



Και τα υπόλοιπα αποτελέσματα:



**AdaBoost**

To AdaBoost θα μπορούσαμε να πούμε ότι χρησιμοποιεί αρκετούς «κακούς» classifiers οι οποίοι λέγονται estimator για να δημιουργήσει έναν ισχυρότερο classifier. Σε κάθε γύρο, το AdaBoost δίνει μεγαλύτερη βαρύτητα στα samples που δεν έγιναν classify σωστά στους προηγούμενους γύρους από τους estimators. Η διαδικασία αυτή επαναλαμβάνεται αρκετές φορές μέχρι να αναπτυχθεί ένας classifier που μπορεί να διαχωρίσει καλά τις κλάσεις.

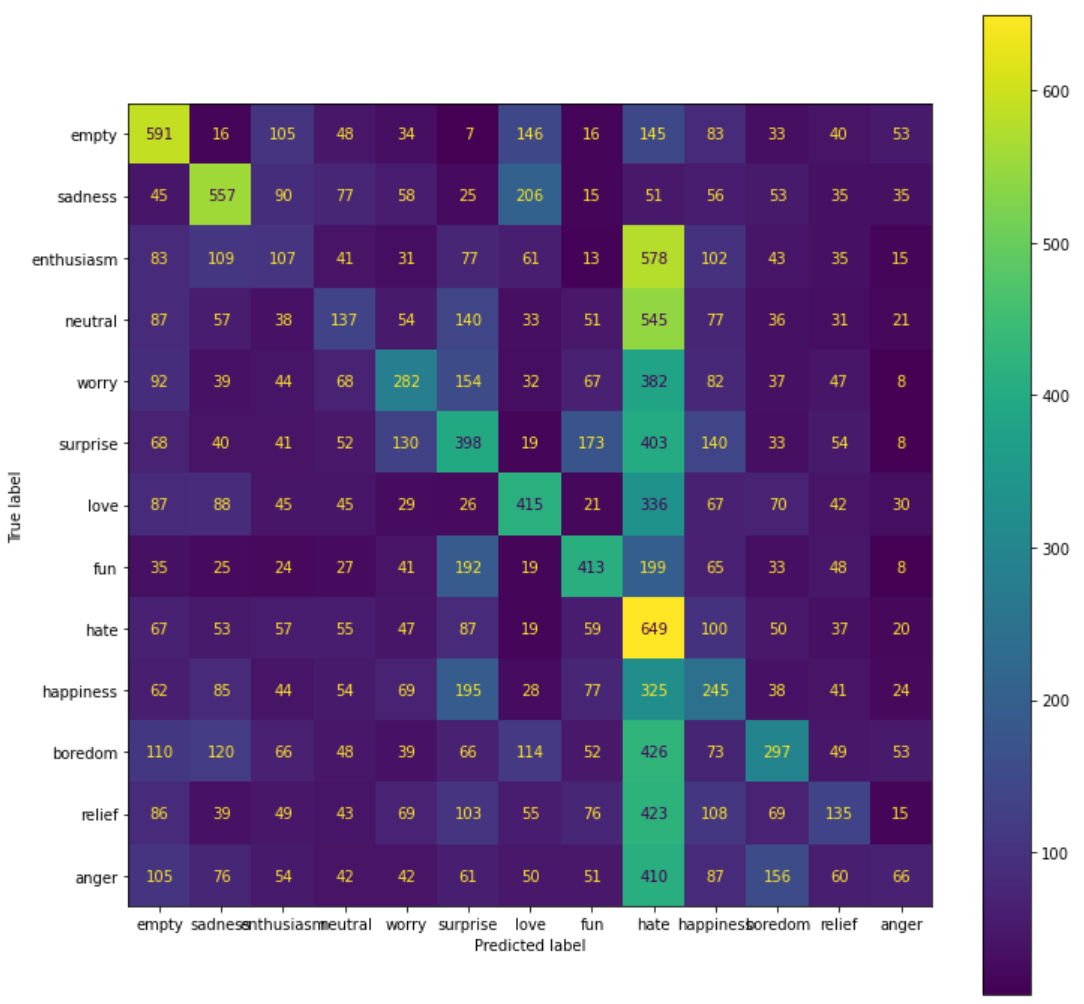
Μία διαφορά του Adaboost από το Random Forest είναι ότι για παράδειγμα στο Random Forest όλα τα Decision Trees έχουν την ίδια βαρύτητα κάτι που δεν ισχύει για τους estimators του Adaboost, καθένας από τους οποίους έχει διαφορετική βαρύτητα.

Τα params που πρέπει να εξεταστούν σε αυτό το μοντέλο είναι το ‘n\_estimators’ που αναφέρεται στον αριθμό των estimators και το ‘learning\_rate’ που επηρεάζει το βάρος του κάθε estimator.

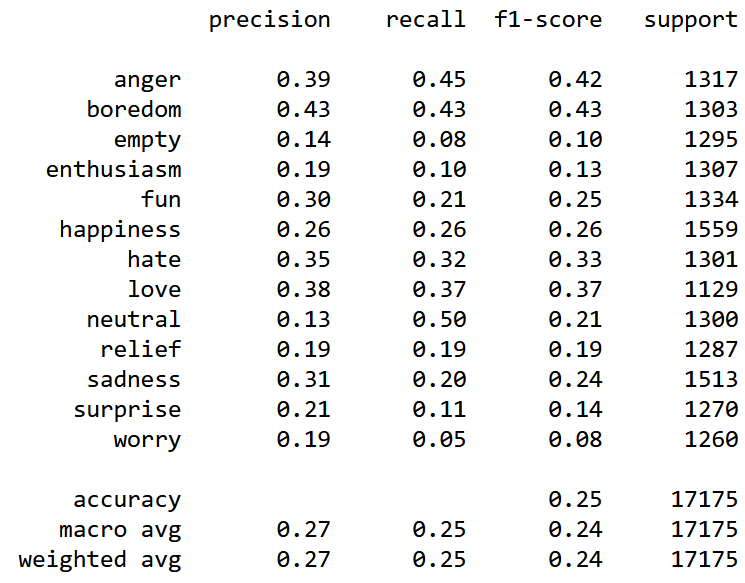
Από το hyperparameter tuning τα καλύτερα params για τα παραπάνω ήταν:



Και το accuracy που πέτυχε το μοντέλο ήταν περίπου 24% στο training και το ίδιο ποσοστό στο validation.



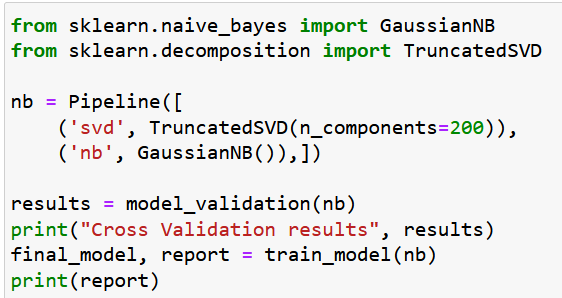
Άλλα στατιστικά στοιχεία:



**Gaussian Naive Bayes**

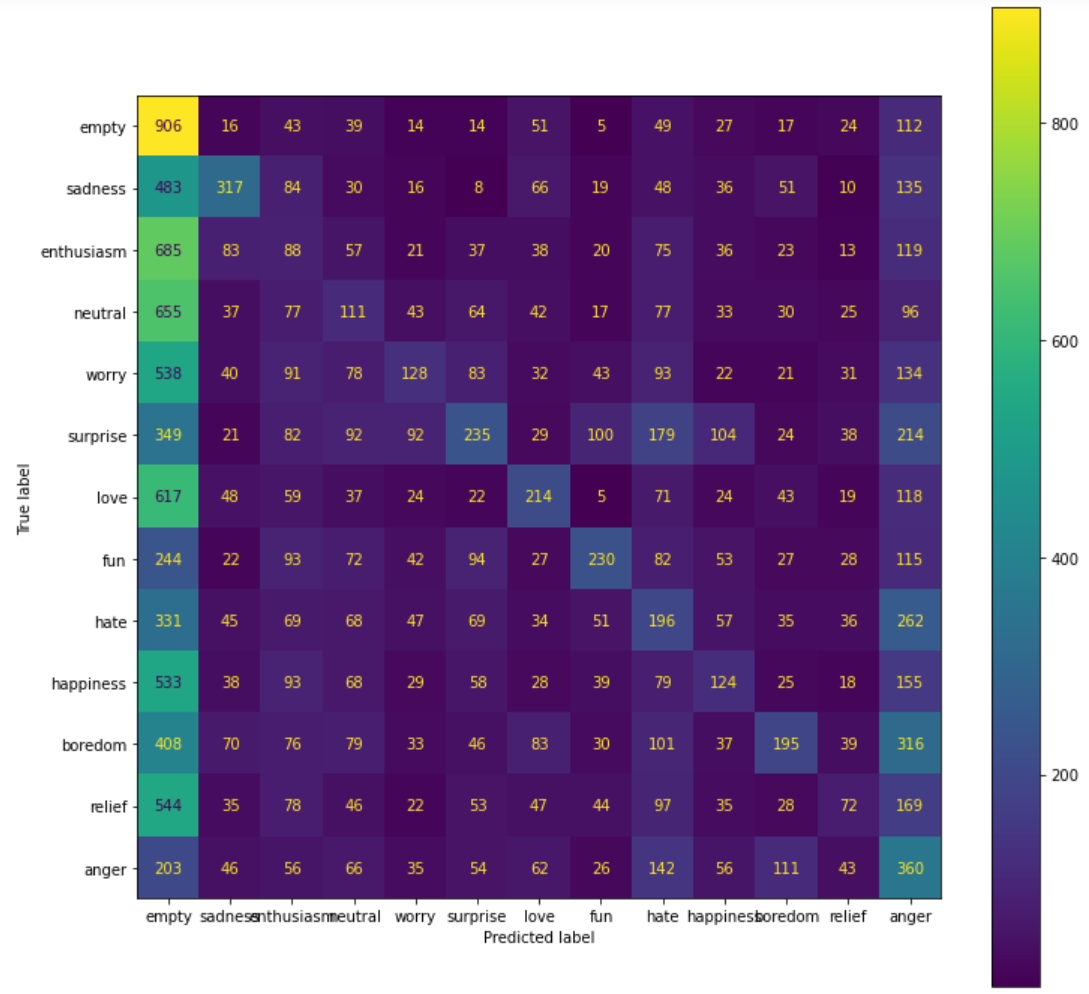
To naive bayes classifier ήταν λίγο διαφορετικό από τα προηγούμενα μοντέλα καθώς έπρεπε να κάνω μερικές τροποποιήσεις στο dataset προκειμένου να λειτουργήσει σωστά. Ο λόγος ήταν ότι καθώς το dataset είναι αρκετά μεγάλο, υπήρχαν αρκετά μηδενικά και επομένως μου έβγαινε συνεχώς error ότι τα δεδομένα είναι ‘too sparse’.

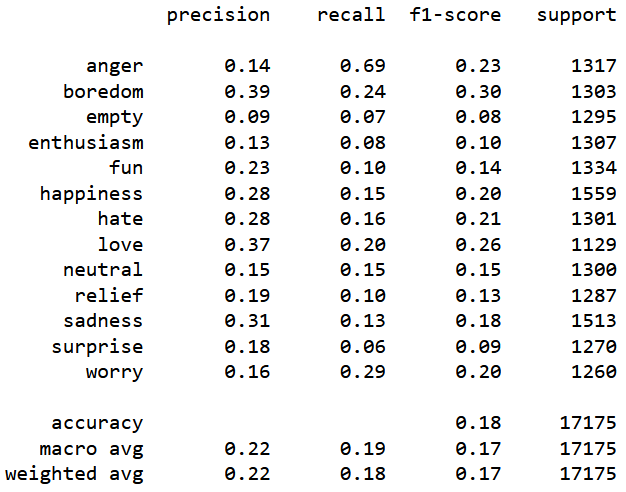
Για την αντιμετώπιση του παραπάνω προβλήματος χρησιμοποίησα την κλάση TruncatedSVD του sklearn έτσι ώστε να μειώσω τα dimensions από 1200 σε 200.



Μετά από αυτή την τροποποίηση πήρα αποτελέσματα και συγκεκριμένα accuracy περίπου 18% στο training και το ίδιο στο validation.

Παραθέτω επίσης τα confusion matrix και τα precision, recall, και f1 κάθε κλάσης:





**Κώδικας Demo και Περαιτέρω Testing**

Από τα παραπάνω μοντέλα, μπορούμε να κάνουμε export όποιο από αυτά επιθυμούμε (μαζί με το vectorizer) και να τρέξουμε τον κώδικα demo.py. Προφανώς, το μέγεθος του μοντέλου μπορεί να διαφοροποιείται ανάλογα με το μοντέλο και τις παραμέτρους εκπαίδευσης που θα επιλέξουμε. Στον κώδικα που βρίσκεται στο repo έχει γίνει export το knn μοντέλο με 3 neighbors από τα παραδείγματα που υπάρχουν στο jupyter notebook.



Το vectorizer και το model αποθηκεύονται στο φάκελο ‘models’.

Ο κώδικας του demo (demo.py) διαβάζει ένα κείμενο από το αρχείο tweet.txt (πρέπει να βρίσκεται στο ίδιο επίπεδο με το demo.py) και επιστρέφει ένα prediction για το sentiment του tweet.

Ο κώδικας περιέχει επίσης τα δύο functions για τη μετατροπή των emojis και τον «καθαρισμό» του tweet από urls, retweet, σύμβολα hashtag κλπ.

Ο κώδικας demo.py τρέχει από το command line με την εντολή: python demo.py

Μερικά ενδεικτικά αποτελέσματα που επιστρέφονται από το demo είναι:

