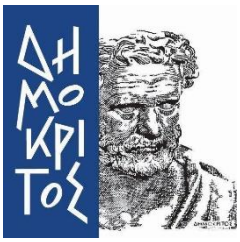
Tweet Sentiment Analysis



Μηχανική Μάθηση

Εργασία Εξαμήνου

Τζανής Νικόλαος
AM: mtn2217

Περιεχόμενα

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”:

```
raw_df = pd.read_csv('../datasets/tweet_emotions.csv')
```

Εξερεύνηση και Επεξεργασία των Δεδομένων

Διαστάσεις του Dataframe

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

```
Length of df 40000, number of columns 3, dimensions (40000, 3), number of elements 120000
```

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:

```
raw_df.info(memory_usage='deep')
```

Το αποτέλεσμα της παραπάνω εντολής:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   tweet_id   40000 non-null  int64
 1   sentiment  40000 non-null  object
 2   content    40000 non-null  object
dtypes: int64(1), object(2)
memory usage: 7.7 MB
```

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.

```
raw_df.describe(include='all')
```

	tweet_id	sentiment	content
count	4.000000e+04	40000	40000
unique	NaN	13	39827
top	NaN	neutral	I just received a mothers day card from my lov...
freq	NaN	8638	14
mean	1.845184e+09	NaN	NaN
std	1.188579e+08	NaN	NaN
min	1.693956e+09	NaN	NaN
25%	1.751431e+09	NaN	NaN
50%	1.855443e+09	NaN	NaN
75%	1.962781e+09	NaN	NaN
max	1.966441e+09	NaN	NaN

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.

```
raw_df.head(5)
```

	tweet_id	sentiment	content
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...

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.

```
print(raw_df.isna().sum())
print("No missing values after all :)")
```

```
tweet_id     0
sentiment    0
content      0
dtype: int64
No missing values after all :)
```

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.

```
df_size = raw_df.shape[0]
raw_df.drop_duplicates(keep=False, inplace=True)
df_size_dup = raw_df.shape[0]
print(f'Duplicate Rows Deleted {df_size - df_size_dup}, Old size: {df_size}, New
```

```
Duplicate Rows Deleted 0, Old size: 40000, New size: 40000
```

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.

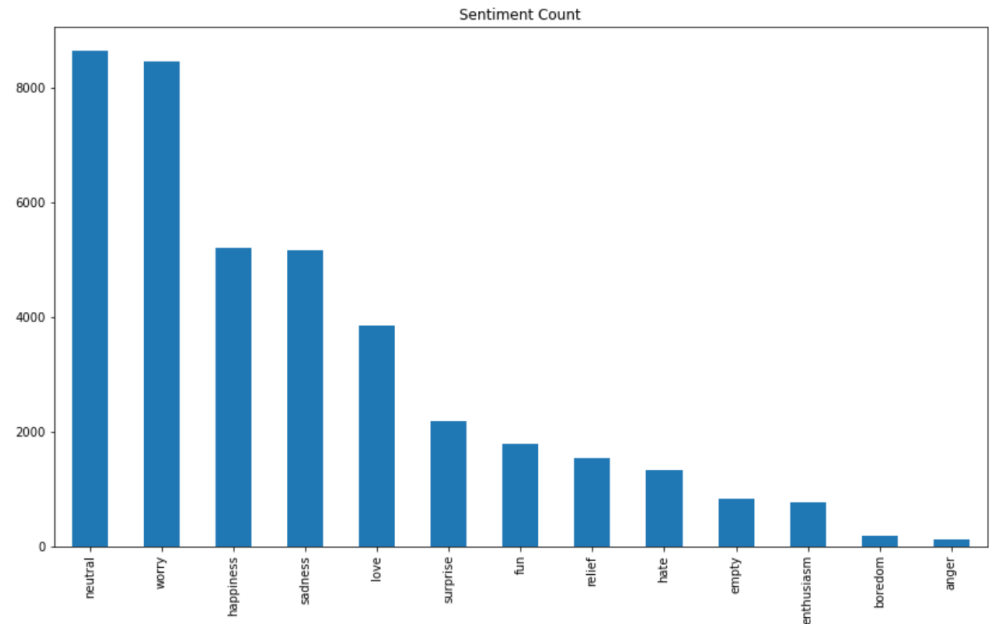
```
print("Sentiment Label has", len(raw_df['sentiment'].unique()),"unique values.")
print(raw_df['sentiment'].value_counts())
```

```
Sentiment Label has 13 unique values.
neutral       8638
worry         8459
happiness     5209
sadness       5165
love          3842
surprise      2187
fun           1776
relief        1526
hate          1323
empty          827
enthusiasm     759
boredom        179
anger          110
Name: sentiment, dtype: int64
```

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:

```
ax = raw_df['sentiment'].value_counts().plot(kind='bar',
                                     figsize=(14,8),
                                     title="Sentiment Count")
```



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'):

```
# Create a column containing the character count
raw_df['content_char_count'] = raw_df['content'].str.len()

# Statistical Insights
raw_df['content_char_count'].describe()
```

```
count    40000.000000
mean        73.405550
std         36.558015
min          1.000000
25%         43.000000
50%         69.000000
75%        103.000000
max        167.000000
Name: content_char_count, dtype: float64
```

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:

```
#  Remove the whole row, for samples that have a character count below 5
old_shape = raw_df.shape[0]
raw_df = raw_df[raw_df['content_char_count'] >= 5]
print(f'Rows Removed for having low number of characters: {old_shape-raw_df.shape[0]}')
```

```
Rows Removed for having low number of characters: 4
```

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

```
plt.figure(figsize=(16,4))
ax = sns.boxplot(x="sentiment",y="content_char_count",data=raw_df, palette="Set3")
plt.show()
```

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:

```
def replace_emoticons(txt):

    # Replace emoticons with a word describing the emoticon
    emoticons_happy = set([':-)', ':)', ':]', ':3', '=]', '=)'])

    emoticons_sad = set([':(',"=[" ,":'(", ':@', ':-(', ':[', ':-[', '>.<', ':-c',':c'])

    emoticons_surprised = set([':O', ':o', ':-o', ':-O'])

    words = txt.split()
    new_w = []
    for w in words:
        if w in emoticons_happy:
            new_w.append("happy")
        elif w in emoticons_sad:
            new_w.append("sad")
        elif w in emoticons_surprised:
            new_w.append("surprised")
        else:
            new_w.append(w)
    txt = " ".join(new_w)
    return txt
```

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:

```
def data_cleaning(txt):
    txt = replace_emoticons(txt)     # replace emoticons with a word describing the emoticon
    txt = re.sub('@[^\\s]+', '', txt) # remove usernames
    txt = re.sub('RT[\\s]+', '', txt) # remove retweet 'RT'
    txt = re.sub('#', '', txt)       # remove '#'
    txt = re.sub('((www\\.[^\\s]+)|(https?://[^\\s]+))', '', txt)  # remove URLs inside the text

    return txt

print('Average word character count BEFORE data cleaning is',raw_df['content_char_count'].mean())
raw_df['content'] = raw_df['content'].apply(data_cleaning)
raw_df['content_char_count'] = raw_df['content'].str.len()
print('Average word character count AFTER data cleaning is',raw_df['content_char_count'].mean())
```

```
Average word character count BEFORE data cleaning is 73.41279127912792
Average word character count AFTER data cleaning is 66.1998699869987
```

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:

```
# Perform the same operation as before and remove rows with Char Count <5
old_shape = raw_df.shape[0]
raw_df = raw_df[raw_df['content_char_count'] >= 5]
print(f'Rows Removed for having low number of characters: {old_shape-raw_df.shape[0]}')
```

```
Rows Removed for having low number of characters: 106
```

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:

```
raw_df = raw_df[['content','sentiment']]
```

```
raw_df.head(5)
```

	content	sentiment
0	i know i was listenin to bad habit earlier an...	empty
1	Layin n bed with a headache ughhhh...waitin on...	sadness
2	Funeral ceremony...gloomy friday...	sadness
3	wants to hang out with friends SOON!	enthusiasm
4	We want to trade with someone who has Houston...	neutral

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.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn import metrics
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('always')

final_vectorizer_params = {
    "smooth_idf": True,
    "analyzer": 'word',
    "stop_words": 'english',
    "lowercase": True,
    "ngram_range": (1,2),
    "max_features": 1200,
}

tfidf_vectorizer = TfidfVectorizer(**final_vectorizer_params)

train_df = raw_df.copy()
print(train_df.shape[0])

X = train_df['content'].copy()
y = train_df['sentiment'].copy()

X = tfidf_vectorizer.fit_transform(X)
```

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.

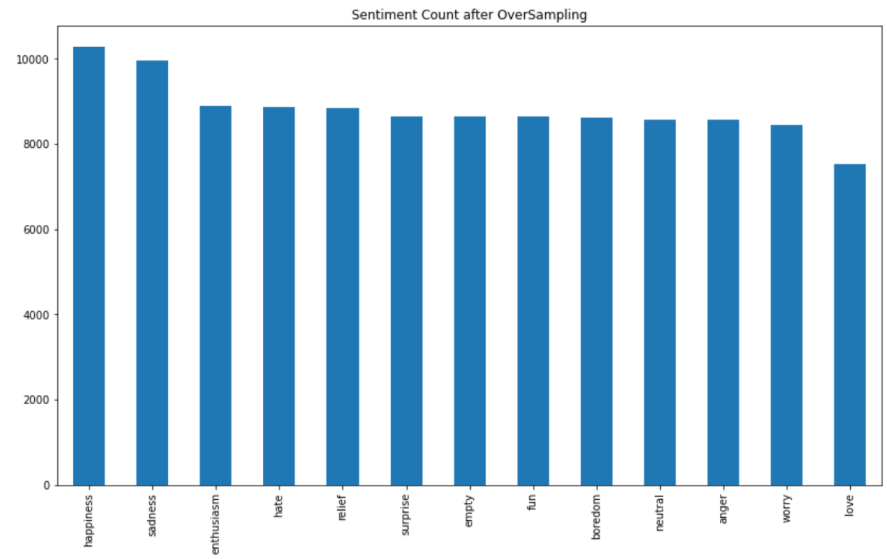
Αναφορικά με τα params της ADASYN, έχω χρησιμοποιήσει το sampling_strategy ‘minority’ 11 φορές γιατί τόσο το ‘majority’ όσο και το ‘auto’ μου έβγαζαν προβλήματα.

```
from imblearn.over_sampling import SMOTE, ADASYN

oversample = ADASYN(sampling_strategy='minority', random_state=8)

for i in range(11):
    X,y = oversample.fit_resample(X, y)

print("New dataset shape",X.shape)
```

```
New dataset shape (114494, 1200)
```

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

```
ax = y.value_counts().plot(kind='bar',
                           figsize=(14,8),
                           title="Sentiment Count after OverSampling")
```



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.

```
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def get_confusion_matrix(y, y_p):
    cm = confusion_matrix(y, y_p)
    fig, ax = plt.subplots(figsize=(12,12))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(raw_df['sentiment'].unique()))
    disp.plot(ax=ax)
    plt.show()
    return

def hyperparameter_tuning(model, params):
    search = GridSearchCV(model, params, n_jobs=-1, verbose=1, cv=2)
    search.fit(X, y)
    print(" Results from Grid Search " )
    print("\n The best estimator across ALL searched params:\n",search.best_estimator_)
    print("\n The best score across ALL searched params:\n",search.best_score_)
    print("\n The best parameters across ALL searched params:\n",search.best_params_)
    return search.best_estimator_

def model_validation(model):
    results = cross_validate(model, X, y, cv=5, scoring=('accuracy', 'precision_macro', 'recall_macro', 'f1_macro'))
    return results

def train_model(model):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, shuffle=True)
    res = model.fit(X_train, y_train)
    predicted = model.predict(X_test)
    result = np.mean(predicted == y_test)
    c_report = classification_report(y_test,predicted)
    print("<split> Accuracy Score is",result)
    get_confusion_matrix(y_test, predicted)
    return res, c_report
```

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).

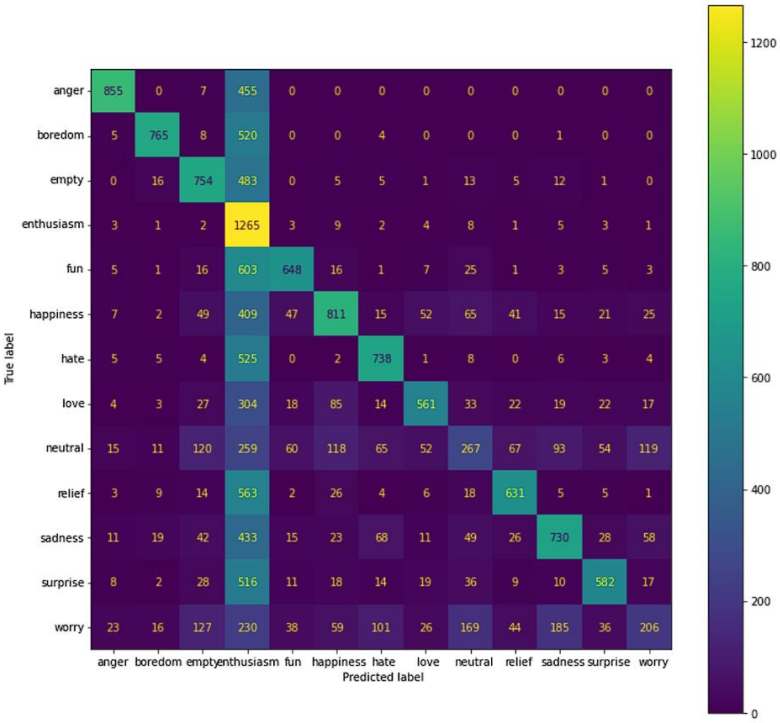
```
from sklearn.svm import SVC

svc = SVC()

param_grid = {
    "kernel": ['linear', 'poly'],
    "C": [0.6, 0.8, 1, 1.2, 1.4],
    "decision_function_shape": ['ovr']
}
best_model = hyperparameter_tuning(svc, param_grid)
results = model_validation(best_model)
print("Cross Validation results", results)
final_model, report = train_model(best_model)
print(report)
```

Τα καλύτερα params τελικά σύμφωνα με το tuning είναι τα:

```
The best parameters across ALL searched params:
{'C': 0.6, 'decision_function_shape': 'ovr', 'kernel': 'poly'}
```

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:



	precision	recall	f1-score	support
anger	0.91	0.65	0.76	1317
boredom	0.90	0.59	0.71	1303
empty	0.63	0.58	0.60	1295
enthusiasm	0.19	0.97	0.32	1307
fun	0.77	0.49	0.60	1334
happiness	0.69	0.52	0.59	1559
hate	0.72	0.57	0.63	1301
love	0.76	0.50	0.60	1129
neutral	0.39	0.21	0.27	1300
relief	0.74	0.49	0.59	1287
sadness	0.67	0.48	0.56	1513
surprise	0.77	0.46	0.57	1270
worry	0.46	0.16	0.24	1260
accuracy			0.51	17175
macro avg	0.66	0.51	0.54	17175
weighted avg	0.66	0.51	0.54	17175

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’.

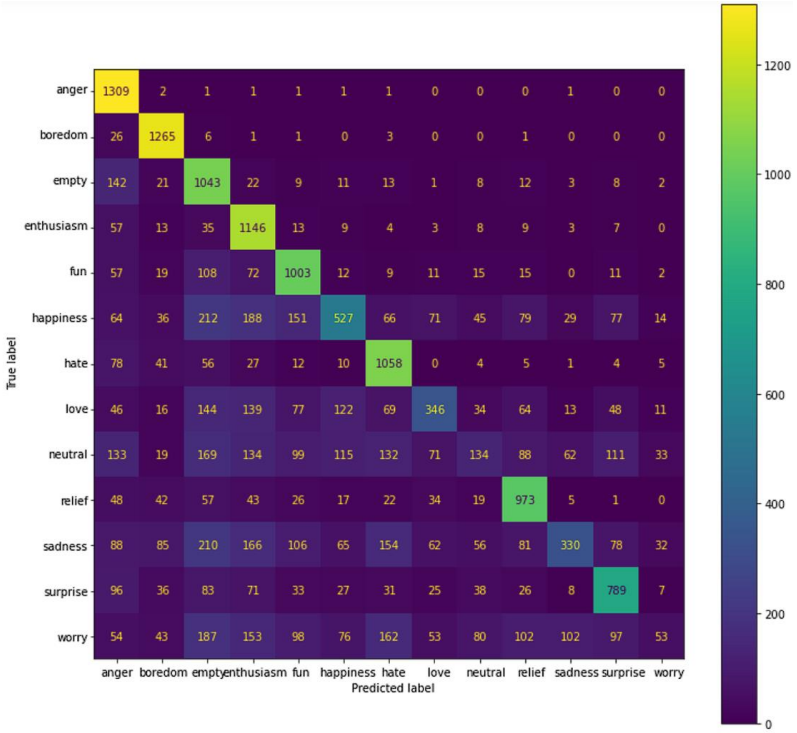
Εδώ προφανώς έχει σημασία πόσα είναι τα γειτονικά data points που θα εξετάσουμε καθώς μία μικρότερη τιμή του ‘n_neighbors’ μπορεί να μας επιτρέψει να αντιληφθούμε πιο σύνθετες συσχετίσεις στα δεδομένα, ενώ μία μεγαλύτερη τιμή μπορεί να δημιουργήσει μεν ένα ισχυρότερο μοντέλο που δεν κάνει overfit, αλλά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

```
The best parameters across ALL searched params:
{'n_neighbors': 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:



	precision	recall	f1-score	support
anger	0.60	0.99	0.74	1317
boredom	0.77	0.97	0.86	1303
empty	0.45	0.81	0.58	1295
enthusiasm	0.53	0.88	0.66	1307
fun	0.62	0.75	0.68	1334
happiness	0.53	0.34	0.41	1559
hate	0.61	0.81	0.70	1301
love	0.51	0.31	0.38	1129
neutral	0.30	0.10	0.15	1300
relief	0.67	0.76	0.71	1287
sadness	0.59	0.22	0.32	1513
surprise	0.64	0.62	0.63	1270
worry	0.33	0.04	0.07	1260
accuracy			0.58	17175
macro avg	0.55	0.58	0.53	17175
weighted avg	0.55	0.58	0.53	17175

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.

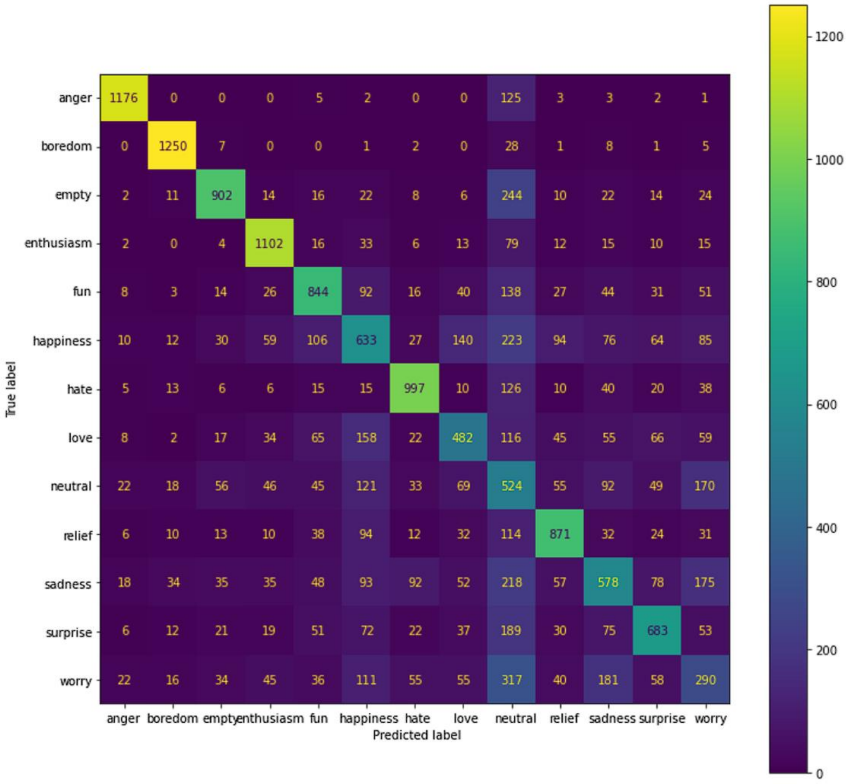
Μία άλλη παράμετρος είναι η ‘min samples leaf’. Οι μικρές τιμές σε αυτό το metric αυξάνουν την πιθανότητα overfit, ενώ σε μεγάλες τιμές υπάρχει πιθανότητα underfit.

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.

```
The best parameters across ALL searched params:
{'max_depth': 500, 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter': 'best'}
```

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

Το confusion matrix ήταν:



Και τα υπόλοιπα δεδομένα:

	precision	recall	f1-score	support
anger	0.92	0.89	0.90	1317
boredom	0.91	0.96	0.93	1303
empty	0.79	0.70	0.74	1295
enthusiasm	0.79	0.84	0.82	1307
fun	0.66	0.63	0.64	1334
happiness	0.44	0.41	0.42	1559
hate	0.77	0.77	0.77	1301
love	0.51	0.43	0.47	1129
neutral	0.21	0.40	0.28	1300
relief	0.69	0.68	0.69	1287
sadness	0.47	0.38	0.42	1513
surprise	0.62	0.54	0.58	1270
worry	0.29	0.23	0.26	1260
accuracy			0.60	17175
macro avg	0.62	0.60	0.61	17175
weighted avg	0.62	0.60	0.61	17175

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.

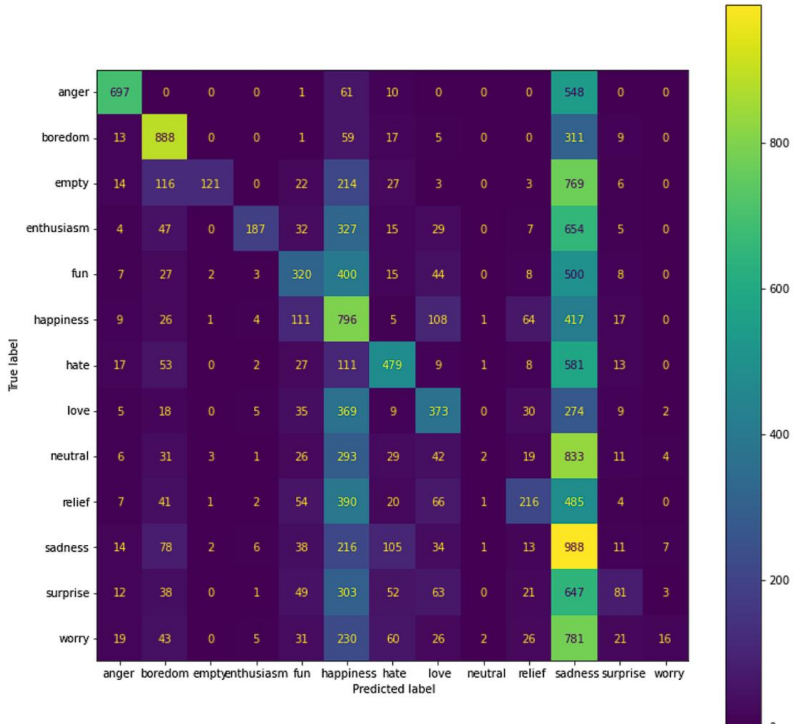
Δοκίμασα τιμές 100 και 200 για το n_estimators και 50, 100 για το max_depth.

Το αποτέλεσμα του hyperparameter tuning ήταν:

```
The best parameters across ALL searched params:
{'criterion': 'entropy', 'max_depth': 100, 'n_estimators': 200}
```

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:

	precision	recall	f1-score	support
anger	0.85	0.53	0.65	1317
boredom	0.63	0.68	0.66	1303
empty	0.93	0.09	0.17	1295
enthusiasm	0.87	0.14	0.25	1307
fun	0.43	0.24	0.31	1334
happiness	0.21	0.51	0.30	1559
hate	0.57	0.37	0.45	1301
love	0.47	0.33	0.39	1129
neutral	0.25	0.00	0.00	1300
relief	0.52	0.17	0.25	1287
sadness	0.13	0.65	0.21	1513
surprise	0.42	0.06	0.11	1270
worry	0.50	0.01	0.02	1260
accuracy			0.30	17175
macro avg	0.52	0.29	0.29	17175
weighted avg	0.51	0.30	0.29	17175

AdaBoost

Το 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.

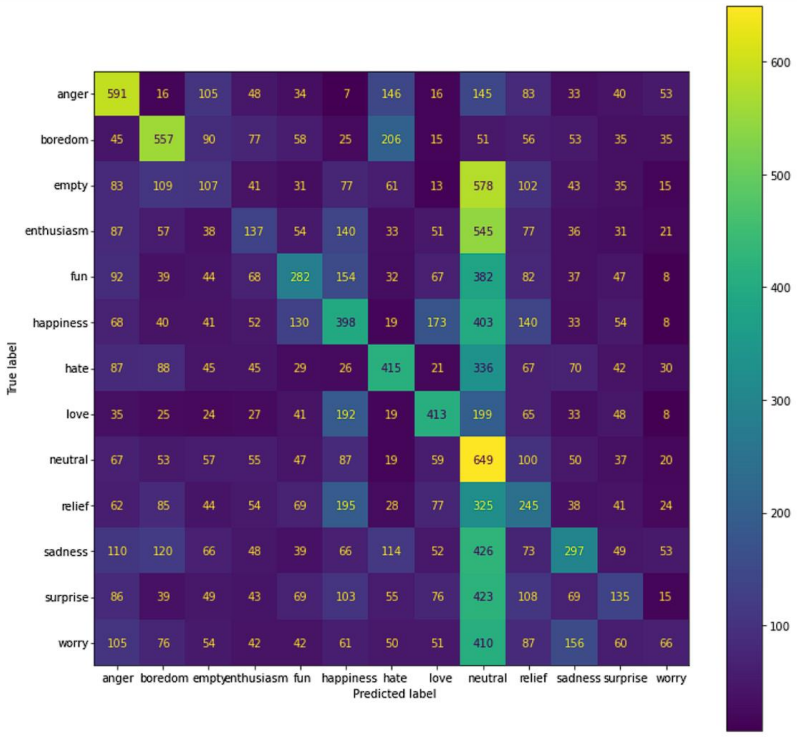
Μία διαφορά του Adaboost από το Random Forest είναι ότι για παράδειγμα στο Random Forest όλα τα Decision Trees έχουν την ίδια βαρύτητα κάτι που δεν ισχύει για τους estimators του Adaboost, καθένας από τους οποίους έχει διαφορετική βαρύτητα.

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:

```
The best parameters across ALL searched params:
{'algorithm': 'SAMME.R', 'learning_rate': 1.0, 'n_estimators': 200}
```

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



Άλλα στατιστικά στοιχεία:

	precision	recall	f1-score	support
anger	0.39	0.45	0.42	1317
boredom	0.43	0.43	0.43	1303
empty	0.14	0.08	0.10	1295
enthusiasm	0.19	0.10	0.13	1307
fun	0.30	0.21	0.25	1334
happiness	0.26	0.26	0.26	1559
hate	0.35	0.32	0.33	1301
love	0.38	0.37	0.37	1129
neutral	0.13	0.50	0.21	1300
relief	0.19	0.19	0.19	1287
sadness	0.31	0.20	0.24	1513
surprise	0.21	0.11	0.14	1270
worry	0.19	0.05	0.08	1260
accuracy			0.25	17175
macro avg	0.27	0.25	0.24	17175
weighted avg	0.27	0.25	0.24	17175

Gaussian Naive Bayes

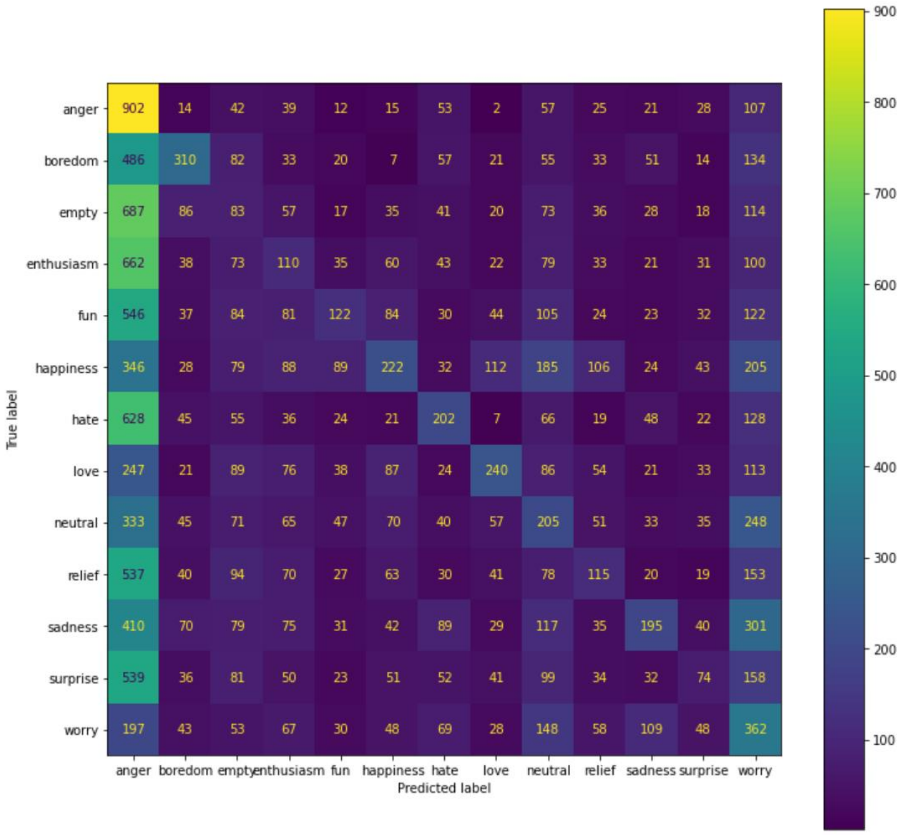
Το naive bayes classifier ήταν λίγο διαφορετικό από τα προηγούμενα μοντέλα καθώς έπρεπε να κάνω μερικές τροποποιήσεις στο dataset προκειμένου να λειτουργήσει σωστά. Ο λόγος ήταν ότι καθώς το dataset είναι αρκετά μεγάλο, υπήρχαν αρκετά μηδενικά και επομένως μου έβγαινε συνεχώς error ότι τα δεδομένα είναι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

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

```
from sklearn.naive_bayes import GaussianNB
from sklearn.decomposition import TruncatedSVD

nb = Pipeline([
    ('svd', TruncatedSVD(n_components=200)),
    ('nb', GaussianNB()),])

results = model_validation(nb)
print("Cross Validation results", results)
final_model, report = train_model(nb)
print(report)
```

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:



	precision	recall	f1-score	support
anger	0.14	0.69	0.23	1317
boredom	0.39	0.24	0.30	1303
empty	0.09	0.07	0.08	1295
enthusiasm	0.13	0.08	0.10	1307
fun	0.23	0.10	0.14	1334
happiness	0.28	0.15	0.20	1559
hate	0.28	0.16	0.21	1301
love	0.37	0.20	0.26	1129
neutral	0.15	0.15	0.15	1300
relief	0.19	0.10	0.13	1287
sadness	0.31	0.13	0.18	1513
surprise	0.18	0.06	0.09	1270
worry	0.16	0.29	0.20	1260
accuracy			0.18	17175
macro avg	0.22	0.19	0.17	17175
weighted avg	0.22	0.18	0.17	17175

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.

```
import pickle
from joblib import dump, load

final_vectorizer_params = {
    "smooth_idf": True,
    "analyzer": 'word',
    "stop_words": 'english',
    "lowercase": True,
    "ngram_range": (1,2),
    "max_features": 1200,
}

tfidf_vectorizer = TfidfVectorizer(**final_vectorizer_params)

tfidf_vectorizer.fit(train_df['content'])

final_model_params = {
    'n_neighbors': 3
}

final_model_knn = KNeighborsClassifier(**final_model_params)

final_model_knn.fit(X,y)
```

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:

```
$ python demo.py
The input text is:  I am sad
The predicted sentiment is: sadness
```

```
$ python demo.py
The input text is:  I don't like this building :(
The predicted sentiment is: boredom
```

```
$ python demo.py
The input text is:  I'm so excited about the new PS5!
The predicted sentiment is: enthusiasm
```

```
$ python demo.py
The input text is:  Didn't think I could be any more disappointed with
my country's criminal justice system. I was wrong.
The predicted sentiment is: sadness
```