ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

ΘΕΩΡΙΑ ΑΠΟΦΑΣΕΩΝ

Sentiment Analysis of Twitter Data

ΦΟΙΒΟΣ ΑΛΛΑΓΙΩΤΗΣ ΑΜ: 1056636

ΑΝΤΡΕΑΣ ΧΑΤΖΗΓΑΒΡΙΗΛ ΑΜ: 1056645

ΗΛΙΑΣ ΚΑΡΥΔΗΣ ΑΜ: 1056921

Περιεχόμενα

1. Εισαγωγή

Στόχος της εργασίας μας είναι η συναισθηματική ανάλυση tweets από το Twitter. Υπάρχουν πολλά σύνολα δεδομένων ανάλυσης συναισθημάτων (sentiment analysis) από το Twitter. Ένα από τα πιο δημοφιλή σύνολα δεδομένων ονομάζεται sentiment 140, το οποίο περιέχει 1,6 εκατομμύρια προ επεξεργασμένα Tweets.

Η διαδικασία που ακολουθήσαμε κατά την διάρκεια της εργασίας μας έχει ως εξής: Αρχικά, το σύνολο δεδομένων έπρεπε να προ επεξεργαστεί. Δηλαδή τα tweets έπρεπε να γίνουν όλα lowercase, να μετατρέψουμε τα emojis από emojis σε χαρακτήρες από γράμματα, να φύγουν τα stop words καθαρίζοντας έτσι το tweet επιπλέον. Παρακάτω, χωρίζεται το dataset σε training και testing σύνολα, όπου το κάθε κομμάτι μετά περνάει σε ένα Vectorizer, ο οποίος μετατρέπει κάθε λέξη του tweet σε ένα αριθμό. Αυτό το βήμα είναι απαραίτητο, καθώς τα μοντέλα μηχανικής μάθησης δεν κατανοούν λέξεις, αλλά μόνο αριθμούς.

Αφού ολοκληρωθεί η προ επεξεργασία αυτών των Tweets και η μετατροπή τους σε διανύσματα, χρησιμοποιούμε διάφορά μοντέλα ταξινόμησης, τα οποία εκπαιδεύονται με βάση το σχετικό συναίσθημα του κάθε tweet. Συγκεκριμένα χρησιμοποιήθηκαν οι αλγόριθμοι, XGBoost, Naive Bayes, Logistic Regression, Decision Tree Classifier.

Επιπλέον, ως ένα περαιτέρω βήμα στην εργασία μας, χρησιμοποιήθηκε ένα μοντέλο βαθιάς μάθησης για να καταλήξουμε σε μια πιο σωστή πρόβλεψη συναισθήματος. Συγκεκριμένα φτιάχτηκε ένα νευρωνικό δίκτυο, με χρήση του αλγόριθμου LSTM, τον οποίο έχουμε ενώσει με άλλα κρυφά επίπεδα ενός νευρωνικού δικτύου.

Τέλος, ενώνουμε το πρόγραμμα μας με ένα account του Twitter και ζητάμε από τον χρήστη ένα hashtag για το οποίο σαρώνουμε το Twitter για να ανακτήσουμε τα αντίστοιχα tweets που ανέβηκαν στο συγκεκριμένο Μέσο Κοινωνικής Δικτύωσης. Επίσης, από τον χρήστη ζητείται και μία ημερομηνία από την οποία ξεκινάει η σάρωση.

2. Dataset

Το σύνολο δεδομένων που χρησιμοποιούμε ονομάζεται, sentiment 140, το οποίο περιέχει 1,6 εκατομμύρια προ επεξεργασμένα Tweets. Αυτά τα Tweets έχουν σχολιαστεί και η μεταβλητή στόχος είναι το συναίσθημα. Οι μοναδικές τιμές σε αυτήν τη στήλη είναι 0 (αρνητικό) και 4 (θετικό). Τα tweets επίσης περιέχουν τις εξής πληροφορίες:

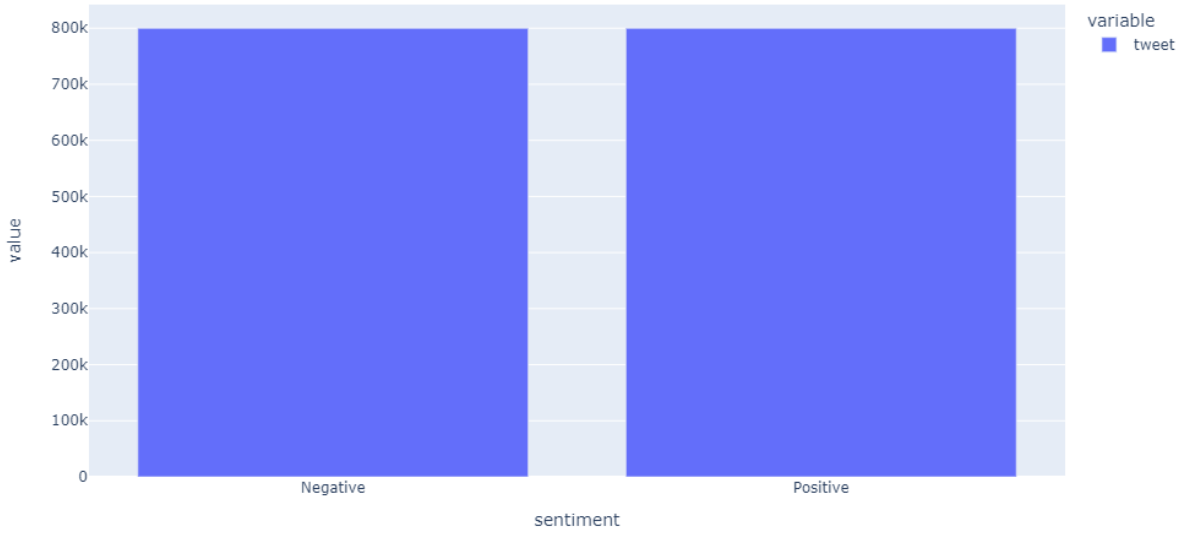
1) target: η πολικότητα του tweet (0 = αρνητικό, 4 = θετικό)
2) ids: Το αναγνωριστικό του tweet (π.χ., 2087)
3) date: την ημερομηνία του tweet (της μορφής, Sat May 16 23:58:44 UTC 2009)
4) flag: Το ερώτημα. Εάν δεν υπάρχει ερώτημα, τότε αυτή η τιμή είναι NO_QUERY.
5) user: ο χρήστης που έκανε tweet (π.χ., robotickilldozr)
6) text: το κείμενο του tweet (π.χ., Lyx is cool)

Αρχικά το σύνολο διαβάζεται ως εξής:

```
# ---------------------------- DATA COLLECTION ---------------------------- #

ENCODING_DATA ="ISO-8859-1"
COLUMN_NAMES = ["sentiment", "ids", "date", "flag", "user", "text"]
dataset = pd.read_csv(r"../Dataset_2.csv", encoding=ENCODING_DATA,
names=COLUMN_NAMES)

# Choose only the columns we want to use
dataset = dataset[['sentiment','text']]
# Replacing the values to ease understanding
dataset['sentiment'] = dataset['sentiment'].replace(4,1)
```

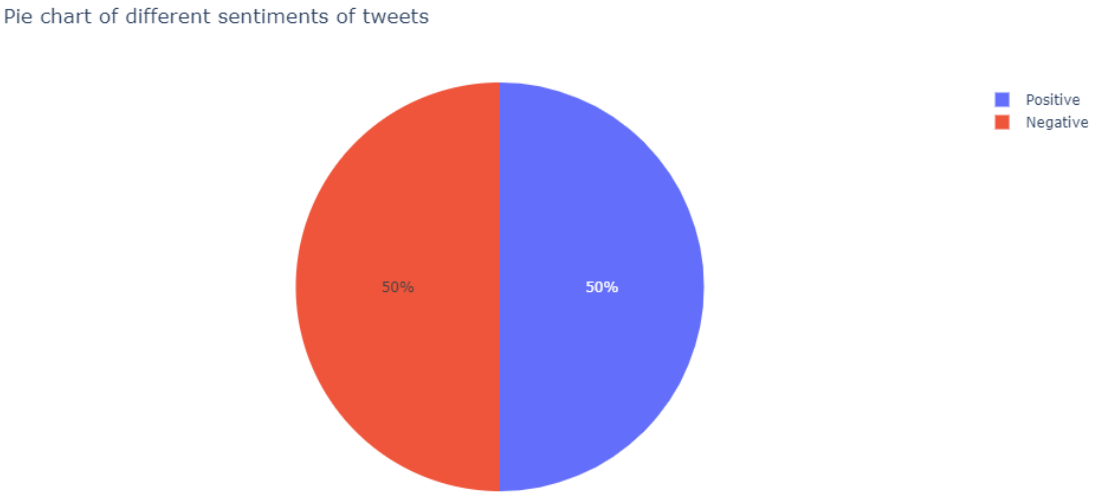
Παρακάτω παραθέτουμε μερικά screenshots που παραπέμπουν σε διάφορες καταστάσεις του συνόλου δεδομένων:

- Αρχικά παραθέτουμε την κατανομή των συναισθημάτων των tweets. Όπως βλέπουμε τα tweets είναι ισοκατανεμημένα μεταξύ των δύο συναισθημάτων.



Διάγραμμα 1. Κατανομή των συναισθημάτων

- Το προηγούμενο διάγραμμα επιβεβαιώνεται και από το ακόλουθο pie γράφημα.

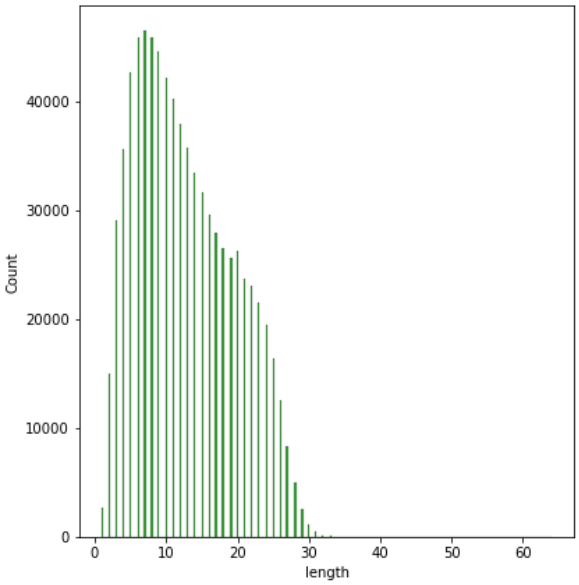


Διάγραμμα 3. Κατανομή των συναισθημάτων σε μορφή πίτας

- Στην συνέχεια, έχουμε την κατανομή βάσει μήκους των tweets που οδηγούν σε θετικό συναίσθημα.

Distribution of text length for positive sentiment tweets.

	length
count	800000.0
mean	12.77
std	6.82
min	1.0
25%	7.0
50%	12.0
75%	18.0
max	64.0

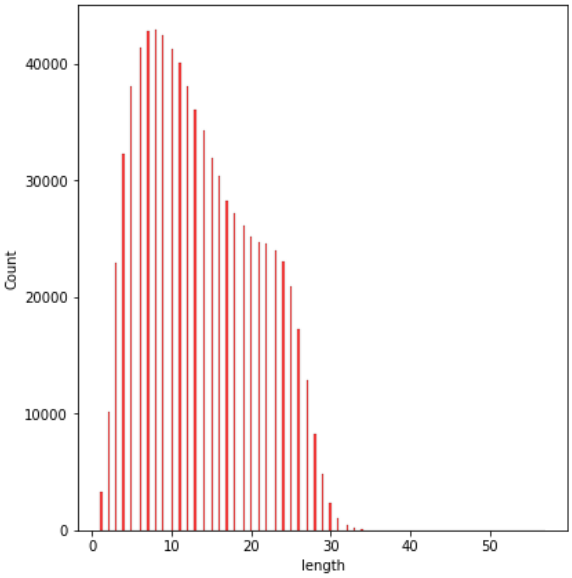


Διάγραμμα 4. Κατανομή μήκους tweets με θετικό συναίσθημα.

- Αντίστοιχα, έχουμε την κατανομή βάσει μήκους των tweets που οδηγούν σε αρνητικό συναίσθημα.

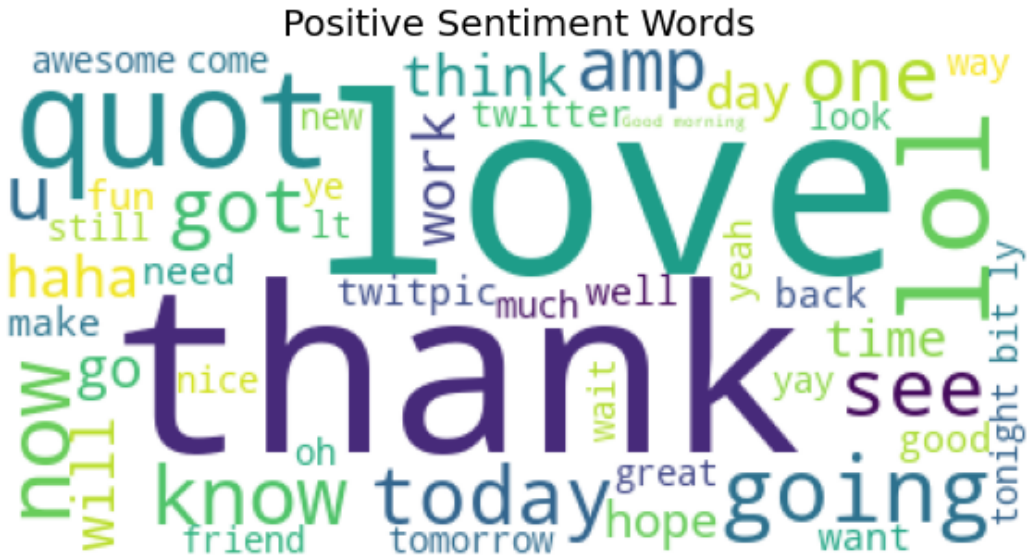
Distribution of text length for Negative sentiment tweets.

	length
count	799999.0
mean	13.58
std	7.07
min	1.0
25%	8.0
50%	13.0
75%	19.0
max	57.0



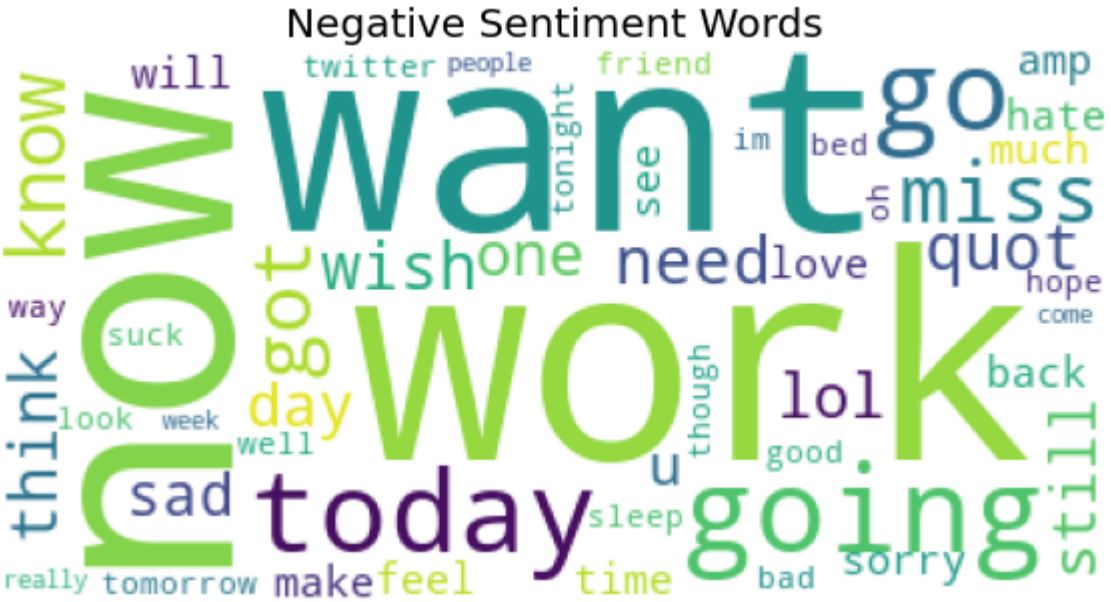
Διάγραμμα 5. Κατανομή μήκους tweets με αρνητικό συναίσθημα.

- Τελευταία, έχουμε την συχνότητα των λέξεων που οδηγούν σε θετικό συναίσθημα, σε μορφή σύννεφο λέξεων.



Διάγραμμα 6. Σύννεφο λέξεων για τις λέξεις των θετικών tweet

- Αντίστοιχα, παραθέτουμε την συχνότητα των λέξεων που οδηγούν σε αρνητικό συναίσθημα, σε μορφή σύννεφο λέξεων.



Διάγραμμα 7. Σύννεφο λέξεων για τις λέξεις των αρνητικών tweet

3. Data Preprocessing

Υπάρχει πολύς θόρυβος στα δεδομένα ακατέργαστου κειμένου που εξάγονται από τα tweets. Τα δύο κρίσιμα μέρη του καθαρισμού κειμένου για την ανάλυση συναισθήματος περιλαμβάνουν: την αφαίρεση των κενών λέξεων και λέξεων που απορρέουν από συγκεκριμένες λέξεις.

Υπάρχουν σημεία στίξης, σύμβολα που δεν θα συμβάλουν πολύ στο μοντέλο μας. Υπάρχουν επίσης κενές λέξεις που πρέπει να αφαιρεθούν. Οι κενές λέξεις αναφέρονται στις συνδετικές λέξεις όπως «το», «και» «ήταν», οι οποίες δεν παρέχουν κάποιο συγκεκριμένο νόημα, κάτι που δεν θα βοηθήσει την ανάλυσή μας. Ως εκ τούτου, τα αφαιρούμε και καθαρίζουμε τα δεδομένα. Το NLTK είναι ένα πακέτο python που χρησιμοποιείται συνήθως για επεξεργασία φυσικής γλώσσας (NLP – Natural Language Processing). Χρησιμοποιώντας αυτό το πακέτο, μπορούμε να λάβουμε γρήγορα όλες τις λέξεις στα αγγλικά.

```
# -------------------- DATA PREPROCESSING/PREPARATION -------------------- #

# Process Time
t = time.time()
# Processed Dataset
processedtext = preprocess(text)
print(f"Text Preprocessing complete.")
print(f"Time Taken: {round(time.time()-t)} seconds \n")
```

Το Stemming/lemmatization αναφέρεται στη διαδικασία εξαγωγής της λέξης ρίζα από μια λέξη. Για παράδειγμα, ένας χρήστης μπορεί να γράψει «παίζω» ως «παίζοντας», «έπαιξε» ή «θα παίξει». Δηλαδή, σε διαφορετικούς χρόνους. Αλλά το πραγματικό νόημα είναι το ίδιο. Πρέπει να τα μετατρέψουμε στη λέξη ρίζα για ευκολότερη μοντελοποίηση. Χρησιμοποιούμε το WordNetLemmatizer από το πακέτο NLTK για να το lemmatization. Για την αφαίρεση των μη αλφαβητικών χαρακτήρων, μπορούμε να χρησιμοποιήσουμε κανονικές εκφράσεις. Το κείμενο μετατρέπεται σε πεζά και αφαιρούνται τα κενά. Οι υπερσύνδεσμοι αφαιρούνται και δημιουργείται μία λίστα για να αποθηκευτούν τα tokens. Η πρόταση χωρίζεται σε λέξεις και κάθε λέξη ελέγχεται αν ανήκει στη λίστα κενών λέξεων. Μετά από αυτό, πραγματοποιείται stemming και η λέξη αποθηκεύεται στη λίστα. Στο τέλος, τα tokens στη λίστα ενώνονται και επιστρέφονται. Όλα τα παραπάνω ορίζονται σε μια συνάρτηση που θα εκτελεί φιλτράρισμα με κανονικές εκφράσεις, εξάγει τις κενές λέξεις και θα εφαρμόζεται σε όλα τα tweets.

```
def preprocess(textdata):
    processedText = []

    # Create Lemmatizer and Stemmer.
    # Lemmatization is the process of converting a word to its base form.
(e.g: “Great” to “Good”)
    wordLemm = WordNetLemmatizer()
```

```
    # Defining regex patterns.
    urlPattern        = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
    userPattern       = '@[^\s]+'
    alphaPattern      = "[^a-zA-Z0-9]"
    sequencePattern   = r"(.)\1\1+"
    seqReplacePattern = r"\1\1"

    for tweet in textdata:

        # Each text is converted to lowercase
        tweet = tweet.lower()

        # Replace all URls with 'URL'
        tweet = re.sub(urlPattern,' URL',tweet)
        # Replace all emojis.
        for emoji in emojis.keys():
            tweet = tweet.replace(emoji, "EMOJI" + emojis[emoji])
        # Replace @USERNAME to 'USER'.
        tweet = re.sub(userPattern,' USER', tweet)
        # Replace all non alphabets.
        tweet = re.sub(alphaPattern, " ", tweet)
        # Replace 3 or more consecutive letters by 2 letter.
        tweet = re.sub(sequencePattern, seqReplacePattern, tweet)

        tweetwords = ''
        for word in tweet.split():
            # Checking if the word is a stopword.
            #if word not in stopwordlist:
            if len(word)>1:
                # Lemmatizing the word.
                word = wordLemm.lemmatize(word)
                tweetwords += (word+' ')

        processedText.append(tweetwords)

    return processedText
```

Η έξοδος έχει ως εξής:

Αρχικό Tweet	Επεξεργασμένο tweet
is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!	['upset', 'update', 'facebook', 'text', 'might', 'cry', 'result', 'school', 'today', 'also', 'blah']

4. Data Splitting

Στην συνέχεια, χωρίζουμε το σύνολο δεδομένων σε σύνολα εκπαίδευσης και δοκιμής. Μπορούμε να το κάνουμε αυτό εύκολα χρησιμοποιώντας τη συνάρτηση “train_test_split()” της βιβλιοθήκης sklearn. Παίρνουμε το 30% του συνόλου δεδομένων για δοκιμαστικούς σκοπούς και το υπόλοιπο για εκπαίδευση.

```
# -------------- DATA SPLITTING TO TRAINING AND TESTING DATA ------------- #

X_train, X_test, y_train, y_test = train_test_split(processedtext,
sentiment, test_size = 0.3, random_state = 0)
print(f'Data Split Complete.')
```

5. Data Tokenization and Vectorization

Το tokenization αναφέρεται στο διαχωρισμό της δεδομένης πρότασης σε μια λίστα με tokens, ευρετηριασμένα ή διανύσματα. Θα χρησιμοποιήσουμε το TfidfVectorizer του sklearn. Οι αλγόριθμοι μηχανικής μάθησης λαμβάνουν αριθμούς ως εισόδους. Αυτό σημαίνει ότι θα χρειαστεί να μετατρέψουμε τα κείμενα σε αριθμητικά διανύσματα που αυτό ουσιαστικά κάνει το TF-IDF Vectorizer. Χρησιμοποιείται όπως φαίνεται παρακάτω

```
# ------------------------- DATA VECTORIZATION ------------------------- #

vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print(f'Vectoriser fitted.')
print('Number of feature words: ', len(vectoriser.get_feature_names()))

X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
print(X_test.shape)
print(X_train.shape)
print(f'Data Transformation to Vectors Completed. \n')
```

Για να καταλάβουμε περισσότερο την λειτουργία αυτής της μεθόδου θα δούμε ένα παράδειγμα:

Ας υποθέσουμε ότι έχετε ένα σύνολο δεδομένων όπου οι μαθητές γράφουν ένα δοκίμιο για το θέμα, “Το σπίτι μου”. Σε αυτό το σύνολο δεδομένων, η λέξη “η” εμφανίζεται πολλές φορές. Δηλαδή είναι λέξη υψηλής συχνότητας σε σύγκριση με άλλες λέξεις στο σύνολο δεδομένων. Το σύνολο δεδομένων περιέχει άλλες λέξεις όπως σπίτι, δωμάτια και ούτω καθεξής που εμφανίζονται λιγότερο συχνά, επομένως η συχνότητα τους είναι χαμηλότερη και μεταφέρουν περισσότερες πληροφορίες σε σύγκριση με τη λέξη. Αυτή είναι η διαίσθηση πίσω από το TF-IDF.

Το πρώτο όρισμα που παίρνει είναι το ngram_range(a, b) το οποίο είναι το εύρος του αριθμού των λέξεων σε μια ακολουθία. Δηλαδή είναι μια σειρά από n-λέξεις στην σειρά. Το a στα ορίσματα του ngram_range είναι το ελάχιστο και το b το μέγιστο μέγεθος n-gram που θέλουμε να συμπεριλάβουμε. Για παράδειγμα “Θεωρία Αποφάσεων” είναι 2-gram, όμως θεωρείτε ότι είναι και 1-gram σπάζοντας τα σε “Θεωρία” και “Αποφάσεων” οπόταν έχουμε n-gram range του (1,2). Στο δικό μας παράδειγμα αποφασίσαμε ότι η μετατροπή σε μονογράμματα λέξεων + διγράμματα παρέχει καλή ακρίβεια, ενώ απαιτεί λιγότερο υπολογιστικό χρόνο. Τέλος σαν δεύτερο όρισμα έχουμε το max_features το οποίο καθορίζει τον αριθμό των χαρακτηριστικών που θέλουμε η μέθοδος του vectoriser να λάβει υπόψη.

Αφού χωρίσουμε τα δείγματα κειμένου μας σε n-gram, πρέπει να μετατρέψουμε αυτά τα n-gram σε αριθμητικά διανύσματα για να μπορούν να επεξεργαστούν στην συνέχεια τα μοντέλα μηχανικής μάθησης για την εκπαίδευση του μοντέλου.

Η έξοδος έχει ως εξής:

Αρχικό Tweet	Επεξεργασμένο tweet
is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!	[8 818 17 111 69 565 193 536 126 2098 9 6 299 551 85 4 2398 149 40 273 1171 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Για το LSTM χρησιμοποιούμε τον CountVectorizer και ο κώδικας έχει ως εξής:

```
# ---------------------- DATA VECTORIZATION FOR LSTM ---------------------- #
count_vector = CountVectorizer(max_features=vocabulary_size,
#                               ngram_range=(1,2),    # unigram and bigram
                              preprocessor=lambda x: x,
                              tokenizer=lambda x: x)

# Fit the training data
X_train = count_vector.fit_transform(X_train).toarray()

# Transform testing data
X_test = count_vector.transform(X_test).toarray()
```

6. Models

Για την εκπαίδευση του συνόλου δεδομένων μας έχουμε χρησιμοποιήσει τα παρακάτω μοντέλα/ αλγορίθμους:

- XGBoost
- Naïve Bayes
- Linear Support Vector Classification (LinearSVC)
- Logistic Regression
- Decision Tree

Πέρα από τους παραπάνω αλγόριθμους, η ομάδα επέλεξε να πειραματιστεί και με μοντέλα βαθιάς μηχανικής μάθησης. Συγκεκριμένα, αποφασίσαμε να χρησιμοποιήσουμε τον αλγόριθμο LSTM, τον οποίο θα ορίσουμε σε παρακάτω κεφάλαιο.

XGBoost

Η τεχνική XGBoost (eXtream Gradient Boosting) είναι ένα εξαιρετικά ευέλικτο εργαλείο που μπορεί να λειτουργήσει μέσω των περισσότερων μορφών παλινδρόμησης και ταξινόμησης προβλημάτων. Το XGBoost έχει μια εξαιρετικά υψηλή προγνωστική δύναμη που το καθιστά την καλύτερη επιλογή για την ακρίβεια στα γεγονότα, καθώς διαθέτει τόσο το γραμμικό μοντέλο όσο και τον αλγόριθμο εκμάθησης δέντρων, κάνοντας τον αλγόριθμο σχεδόν 10 φορές πιο γρήγορο από τις υπάρχουσες τεχνικές ενίσχυσης κλίσης. Αυτό συμβαίνει αντιμετωπίζοντας την πιθανή απώλεια για όλους τους πιθανούς διαχωρισμούς δημιουργίας ενός νέου κλάδου (ειδικά εάν εξετάζουμε την περίπτωση όπου υπάρχουν χιλιάδες χαρακτηριστικά και επομένως χιλιάδες πιθανοί διαχωρισμοί). Το XGBoost αντιμετωπίζει αυτήν την αναποτελεσματικότητα εξετάζοντας τη διανομή των χαρακτηριστικών σε όλα τα δεδομένα σε ένα φύλλο και χρησιμοποιεί αυτές τις πληροφορίες για να μειώσει τον χώρο αναζήτησης πιθανών χωρισμάτων των χαρακτηριστικών. Δηλαδή η XGBoost ελαχιστοποιεί μια κανονικοποιημένη συνάρτηση που συνδυάζει μια κυρτή συνάρτηση απώλειας (με βάση τη διαφορά μεταξύ των προβλεπόμενων και στοχευόντων εξόδων) και έναν όρο ποινής για την πολυπλοκότητα του μοντέλου (με άλλα λόγια, τις συναρτήσεις δέντρου παλινδρόμησης). Η εκπαίδευση συνεχίζεται επαναληπτικά, προσθέτοντας νέα δέντρα που προβλέπουν τα υπολείμματα ή τα σφάλματα προηγούμενων δέντρων που στη συνέχεια συνδυάζονται με προηγούμενα δέντρα για να γίνει η τελική πρόβλεψη.

Naïve Bayes

Ο κανόνας του Bayes επιτρέπει τον υπολογισμό της εκ των υστέρων πιθανότητας από την εκ των προτέρων πιθανότητα, την πιθανοφάνεια και τις αποδείξεις.

Η λειτουργία των αλγορίθμων κατά Bayes βασίζεται στην υπόθεση κατά την οποία η υπό εκμάθηση έννοια συσχετίζεται άμεσα με την κατανομή των πιθανοτήτων που

παρουσιάζουν τα στιγμιότυπα του προβλήματος, όσον αφορά στην κλάση στην οποία ανήκουν.

Στη μάθηση κατά Bayes κάθε παράδειγμα εκπαίδευσης μπορεί σταδιακά να μειώνει ή να αυξάνει την πιθανότητα να είναι σωστή μια υπόθεση. Η δυσκολία στην εφαρμογή της μάθησης κατά Bayes είναι η απαίτηση της γνώσης πολλών τιμών πιθανοτήτων, όταν αυτές οι τιμές δεν είναι δυνατό να υπολογιστούν ακριβώς και υπολογίζονται κατ' εκτίμηση από παλαιότερες υποθέσεις ή εμπειρική γνώση.

Ο ταξινομητής Multinomial Naïve Bayes ταξινομεί ένα γεγονός x_i ($x_i \in \{x_1, x_2, \dots, x_n\}$: χαρακτηριστικά των στιγμιότυπων) σε μία κλάση, η οποία έχει τη μεγαλύτερη πιθανότητα, χρησιμοποιώντας τον κανόνα Bayes:

$$P(y|x_i) = \frac{P(y)P(x_i|y)}{P(x_i)} \quad (1)$$

Τα $P(x_i|y)$ υπολογίζονται από το training set, και εκφράζουν την πιθανότητα εμφάνισης ενός γεγονότος x_i σε μία κλάση y, από το σύνολο των προκαθορισμένων κλάσεων, και υπολογίζεται από τον παρακάτω τύπο:

$$P(x_i|y) = \frac{x_i}{\sum_{i=1}^{n} x_i} \quad (2)$$

Στην περίπτωση που ένα γεγονός δεν υπάρχει στο training set, τότε η πιθανότητα $P(y|x_1, x_2, \dots, x_n)$ μηδενίζεται, αν της αναθέσουμε το στιγμιότυπο του προβλήματος που περιέχει το γεγονός αυτό. Για να αποφευχθούν τέτοιου είδους προβλήματα, γίνεται προσθήκη μίας τιμής σε όλες τις πιθανότητες, με ώστε να μην είναι δυνατός ο μηδενισμός τους.

SVM (Support Vector Machine) Classifier Model

Οι Μηχανές Υποστήριξης Διανυσμάτων (Support Vector Machines, SVMs) είναι ένας αλγόριθμος μηχανικής μάθησης ο οποίος στηρίζεται στη Θεωρία Στατιστικής Μάθησης και στα νευρωνικά δίκτυα τύπου Perceptron. Οι SVM αποτελούν συγγενή των νευρωνικών δικτύων και αντιμετωπίζουν το πρόβλημα της κατηγοριοποίησης επιλέγοντας τα διανύσματα υποστήριξης (support vectors) τα οποία συνορεύουν στο χώρο του προβλήματος με δεδομένα άλλων κλάσεων. Με βάση τα επιλεγμένα αυτά, τα δεδομένα χρησιμοποιούνται για την κατασκευή μιας γραμμικής συνάρτησης διάκρισης (discriminant function), με σκοπό να τα διαχωρίσει όσο το δυνατόν περισσότερο. Στην απλή περίπτωση των 2 διαστάσεων ο αλγόριθμος θα προσπαθήσει να βρει το βέλτιστο υπερεπίπεδο μίας διάστασης, δηλαδή μία γραμμή.

Στην περίπτωση της ταξινόμησης, οι SVM ψάχνουν μια υπερεπιφάνεια (hypersurface) η οποία θα διαχωρίζει τα αρνητικά από τα θετικά παραδείγματα-δεδομένα στο χώρο των παραδειγμάτων. Η υπερεπιφάνεια αυτή επιλέγεται έτσι ώστε να απέχει όσο το δυνατόν περισσότερο από τα κοντινότερα θετικά και αρνητικά παραδείγματα (maximum margin hypersurface). Οι SVM χρησιμοποιούνται ιδιαίτερα για την επίλυση προβλημάτων μάθησης που δεν μπορούν να αντιμετωπιστούν με γραμμικά μοντέλα, επειδή μπορούν να παράγουν μη γραμμικές επιφάνειες απόφασης. Ένα ακόμα πλεονέκτημα των SVM είναι η ικανότητά τους να χειρίζονται πολύ μεγάλους χώρους χαρακτηριστικών. Για τα προβλήματα τα οποία δεν είναι γραμμικά διαχωρίσιμα, μπορούν να χρησιμοποιηθούν μέθοδοι πυρήνων που μετασχηματίζουν ένα μη γραμμικό χώρο εισόδου σε ένα γραμμικό χώρο χαρακτηριστικών.

Σημαντική επίσης είναι και η ανεκτικότητα που παρουσιάζουν οι αλγόριθμοι αυτοί όσον αφορά στο πλήθος των στιγμιότυπων εκπαίδευσης, ιδιαίτερα όταν αυτό διαφέρει μεταξύ των δύο κλάσεων. Αυτό γιατί οι SVM δεν επιδιώκουν να ελαχιστοποιήσουν το σφάλμα των δεδομένων εκπαίδευσης, αλλά να τα διαχωρίσουν αποτελεσματικά σε ένα χώρο μεγάλης διάστασης.

Logistic Regression Model

Η λογιστική παλινδρόμηση (Logistic regression) αποτελεί στην ουσία ένα μοντέλο ταξινόμησης των τιμών μιας μεταβλητής απόκρισης Υ με βάση τη θεωρία των πιθανοτήτων. Ωστόσο μπορεί να χρησιμοποιηθεί τόσο για ταξινόμηση όσο και για προβλήματα παλινδρόμησης, αλλά κυρίως για προβλήματα ταξινόμησης. Στο μοντέλο αυτό όπου η μεταβλητή Υ συνήθως έχει δυαδικό χαρακτήρα (λαμβάνει δύο τιμές) στοχεύετε η πρόβλεψη της έκβασης αυτής από ένα πλήθος προβλεπτικών μεταβλητών που μπορεί να είναι ονομαστικές, τακτικές ή ποσοτικές. Η λογιστική Παλινδρόμηση χρησιμοποιείτε για την πρόβλεψη της κατηγορηματικής εξαρτώμενης μεταβλητής με τη βοήθεια των ανεξάρτητων μεταβλητών.

Διακρίνονται τρεις τύποι λογιστικής παλινδρόμησης ανάλογα με την ιδιαίτερη φύση της εξαρτημένης κατηγορικής μεταβλητής η οποία μπορεί να είναι:

1. Δίτιμη ή δυαδική ή διχοτομική (binary) ή διμερής εξαρτημένη μεταβλητή. Συνίσταται από δύο κατηγορίες, όπως π.χ. είναι οι εκβάσεις επιτυχία/αποτυχία, ΝΑΙ/ΟΧΙ, γεγονός απόν/παρόν.
2. Τακτική (ordinal) μεταβλητή. Η εξαρτημένη μεταβλητή συνίσταται από τρεις ή περισσότερες κατηγορίες μεταξύ των οποίων ισχύει η έννοια της ανισότητας, όπως π.χ. σε μια ερώτηση της κλίμακας διαφωνώ καθόλου, λίγο, μέτρια, αρκετά, πολύ, στην κατάταξη ενός στρώματος υλικού ως λεπτού, μεσαίου, παχέος.
3. Ονομαστική (Nominal) ή πολυωνυμική (polynomial) ή κατηγορική αδιαβάθμητη (non-ordered categorical) ή πολυμερής μεταβλητή απόκρισης.

Decision Tree

Ένα δέντρο απόφασης είναι μια δομή δέντρου που μοιάζει με διάγραμμα ροής όπου ένας εσωτερικός κόμβος (internal node) αντιπροσωπεύει ένα χαρακτηριστικό, το κάθε κλαδί (branch) αντιπροσωπεύει έναν κανόνα απόφασης (decision rule) και κάθε κόμβος φύλλου (leaf node) αντιπροσωπεύει το αποτέλεσμα. Ο κορυφαίος κόμβος σε ένα δέντρο αποφάσεων είναι γνωστός ως κόμβος ρίζας. Μαθαίνει να χωρίζει με βάση την τιμή του χαρακτηριστικού. Διαχωρίζει το δέντρο με αναδρομικό τρόπο κλήσης αναδρομικής κατάτμησης. Αυτή η δομή που μοιάζει με διάγραμμα ροής βοηθά στη λήψη αποφάσεων. Είναι οπτικοποίηση σαν ένα διάγραμμα ροής που μιμείται εύκολα τη σκέψη σε ανθρώπινο επίπεδο. Αυτός είναι ο λόγος για τον οποίο τα δέντρα αποφάσεων είναι εύκολο να κατανοηθούν και να ερμηνευτούν.

7. Data Training

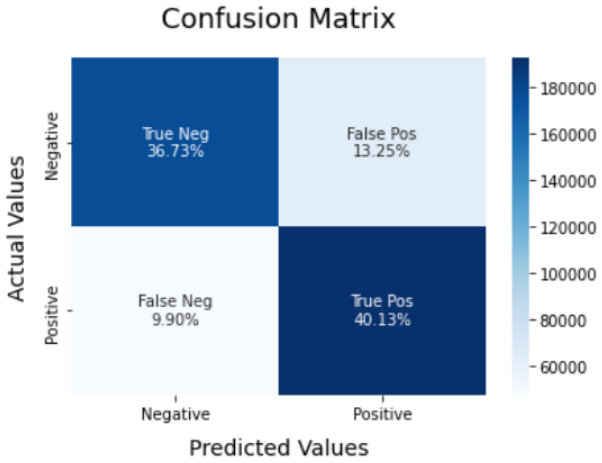
Αρχικά ορίζουμε μία συνάρτηση που είναι υπεύθυνη να μας δείχνει τα στατιστικά ενός μοντέλου καθώς και το confusion matrix του. Η συνάρτηση έχει ως εξής:

```
# -------------------- MODEL ANALYSIS AND EVALUATION -------------------- #

def model_Evaluate(model):

    # Predict values for Test dataset
    y_pred = model.predict(X_test)

    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))

    # Compute and plot the Confusion matrix
    cf_matrix = confusion_matrix(y_test, y_pred)

    categories  = ['Negative','Positive']
    group_names = ['True Neg','False Pos', 'False Neg','True Pos']
    group_percentages = ['{0:.2%}'.format(value) for value in
cf_matrix.flatten() / np.sum(cf_matrix)]

    labels = [f'{v1}\n{v2}' for v1, v2 in
zip(group_names,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    sns.heatmap(cf_matrix, annot = labels, cmap = 'Blues',fmt = '',
xticklabels = categories, yticklabels = categories)

    plt.xlabel("Predicted Values", fontdict = {'size':14}, labelpad = 10)
    plt.ylabel("Actual Values"   , fontdict = {'size':14}, labelpad = 10)
    plt.title ("Confusion Matrix", fontdict = {'size':18}, pad = 20)

    plt.show()

    print('Model Accuracy :',accuracy_score(y_test,y_pred))
```

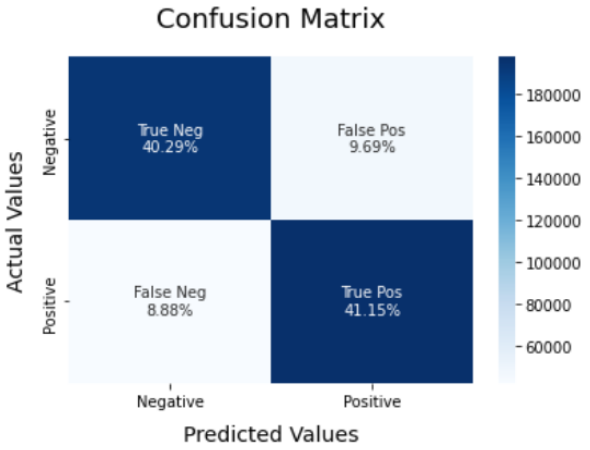
Δεδομένου ότι το σύνολο δεδομένων μας έχει ίσο αριθμό θετικών και αρνητικών προβλέψεων, επιλέγουμε την ακρίβεια (Accuracy) ως μέτρο αξιολόγησης (Metric Evaluation). Επίσης σχεδιάζουμε το Confusion Matrix για κάθε μοντέλο ξεχωριστά για να κατανοήσουμε την απόδοση των μοντέλων που εκπαιδεύουμε.

Τα confusion matrices φαίνονται παρακάτω:

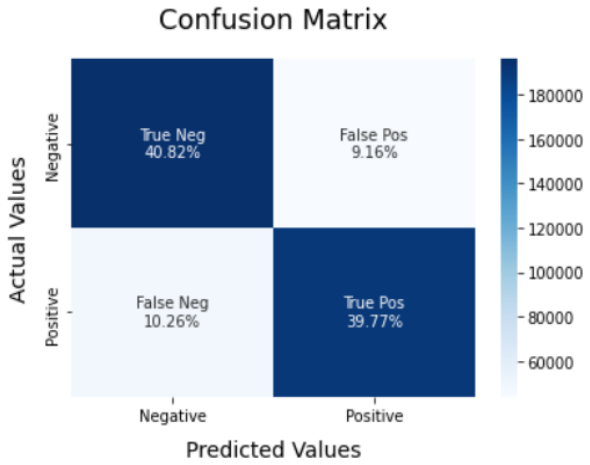
XGBoost: Accuracy = 76,8%



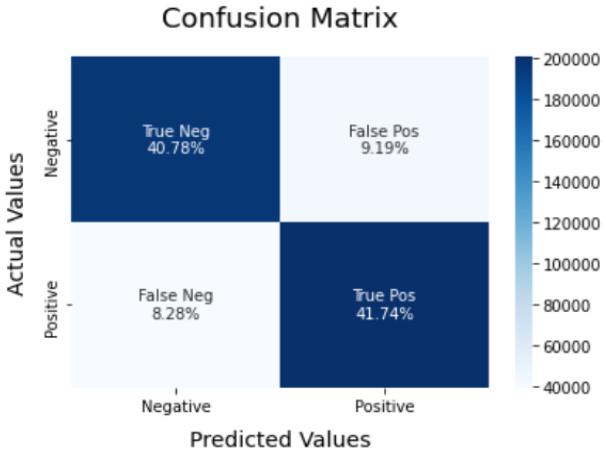
LinearSVC: Accuracy = 81,43%



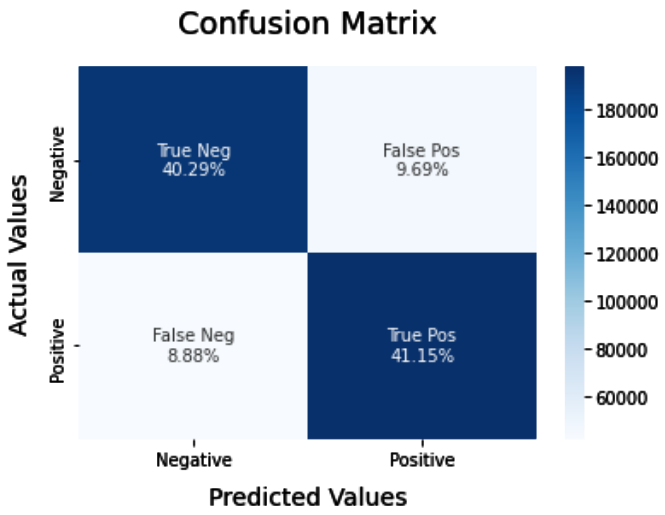
Naïve Bayes: Accuracy = 80,5%



Logistic Regression: Accuracy = 82,5%



Decision Tree: Accuracy = 59,3%



Από το αποτελέσματα που πήραμε μπορούμε να δούμε ότι το μοντέλο Λογιστικής Παλινδρόμησης (Logistic Regression) έχει την καλύτερη απόδοση από όλα τα μοντέλα που έχουμε δοκιμάσει με ακρίβεια σχεδόν 83%.

Ο συνολικός κώδικας φαίνεται παρακάτω:

```
# XGBoost Classifier
xgb_clf = xgb.XGBClassifier(eval_metric='mlogloss',use_label_encoder=False)
xgb_clf.fit(X_train,y_train)
model_Evaluate(xgb_clf)

# Naive Bayes Classifier
nb_clf = MultinomialNB()
nb_clf.fit(X_train,y_train)
model_Evaluate(nb_clf)

# SVM Classifier
SVCmodel = LinearSVC()
SVCmodel.fit(X_train, y_train)
model_Evaluate(SVCmodel)

# Logistic Regrassion
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)

# Decision Tree Classifier
dtc_clf = DecisionTreeClassifier(criterion='entropy', max_depth=2,
random_state=0)
dtc_clf.fit(X_train,y_train)
model_Evaluate(dtc_clf)
```

8. Long Short Term Memory (LSTM)

Δημιουργούμε μια συνάρτηση που μας επιτρέπει να ενωθούμε με το twitter και να εξάγουμε tweets, βάση ενός δοθέν hashtag και μίας ημερομηνίας. Αρχικά, πρέπει να φτιάξουμε ένα λογαριασμό στο twitter και να συμπληρώσουμε συγκεκριμένες στοιχεία έτσι ώστε να μπορούμε να ενωθούμε σαν developers στο twitter. Αφού μας δώσουν τα credentials μας, μπορούμε να κάνουμε αναζήτηση στο Twitter για πρόσφατα tweets. Θα χρησιμοποιήσουμε ένα Cursor για να δημιουργήσουμε ένα αντικείμενο που θα κρατάει τα tweets που ανήκουν στο δοθέν hashtag. Για να δημιουργήσουμε ένα οποιοδήποτε ερώτημα, πρέπει να ορίσουμε: τον όρος αναζήτησης (το hashtag) και την ημερομηνία έναρξης της αναζήτησης. Αφού γίνει η αναζήτηση, περνάμε από όλα τα tweets που αποθηκεύτηκαν στον Cursor και τα κάνουμε append σε μία λίστα. Τέλος, την λίστα αυτή την αποθηκεύουμε σε ένα αρχείο csv.

Το LSTM ή αλλιώς Long Short Term Memory είναι μια τροποποιημένη και προηγμένη αρχιτεκτονική των RNN (Recurrent Neural Networks). Είναι κυρίως χρήσιμο σε διαδοχικά προβλήματα του NLP, όπου το RNN αποτυγχάνει. Τα LSTM είναι ικανά για μοντελοποιήσεις μεγάλης εμβέλειας με καλύτερη ακρίβεια από τα συμβατικά δίκτυα.

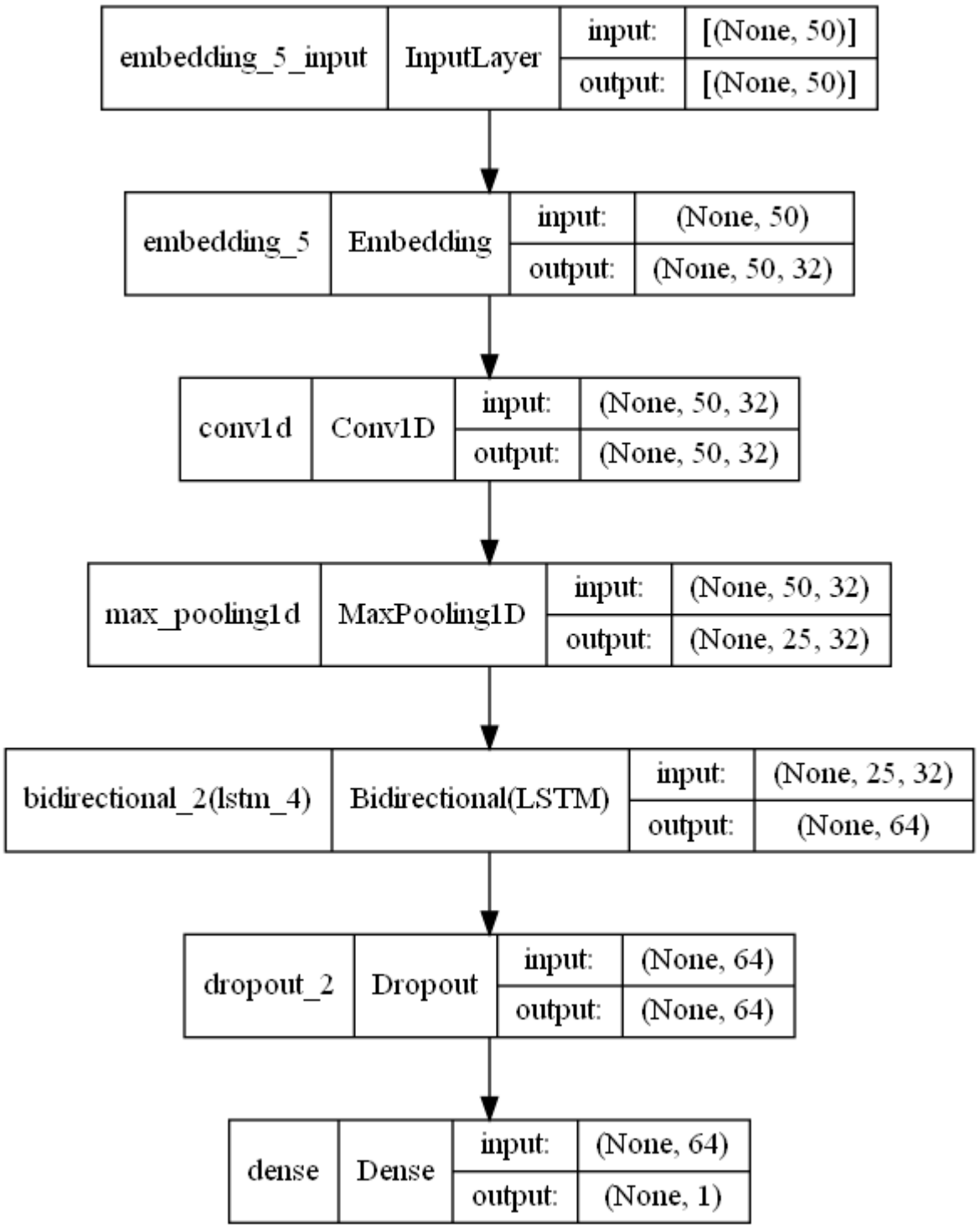
Τα LSTM ασχολούνται τόσο με τη μακροπρόθεσμη μνήμη (Long Term Memory - LTM) όσο και με τη βραχυπρόθεσμη μνήμη (Short Term Memory - STM) και για να κάνει τους υπολογισμούς απλούς και αποτελεσματικούς χρησιμοποιεί την έννοια των πυλών.

Η κύρια αρχιτεκτονική του LSTM έχει ως εξής:

- Forget Gate: Το LTM πηγαίνει στο Forget Gate και ξεχνά πληροφορίες που δεν είναι χρήσιμες.
- Learn Gate: Το συμβάν (τρέχουσα είσοδος) και το STM συνδυάζονται μεταξύ τους, έτσι ώστε οι απαραίτητες πληροφορίες που έμαθε, πρόσφατα το μοντέλο, από το STM να μπορούν να εφαρμοστούν στην τρέχουσα είσοδο.
- Remember Gate: Οι πληροφορίες LTM (που δεν έχουν ξεχαστεί), το STM και το συμβάν συνδυάζονται μαζί στην πύλη μνήμης που λειτουργεί ως ενημερωμένο LTM.
- Use Gate: Αυτή η πύλη χρησιμοποιεί επίσης LTM, STM και το τρέχον συμβάν για να προβλέψει την έξοδο του τρέχοντος συμβάντος που λειτουργεί ως ενημερωμένο STM.

Σε αυτό το πρόβλημα, η αρχιτεκτονική μας αποτελείται από τέσσερα κύρια μέρη. Ξεκινάμε με ένα Embedding Layer, που εισάγει τις ακολουθίες και δίνει ενσωματώσεις λέξεων. Αυτές οι ενσωματώσεις μεταβιβάζονται στη συνέχεια στα Convolution Layers, τα οποία θα τις μετατρέψουν σε μικρά διανύσματα χαρακτηριστικών. Στη συνέχεια, έχουμε το αμφίδρομο LSTM layer. Μετά το LSTM layer, έχουμε ένα Dropout Layer, το οποίο ορίζει τυχαία τις μονάδες εισόδου στο 0 βάση της συχνότητας εμφάνισης των χαρακτηριστικών, σε κάθε βήμα κατά τη διάρκεια του χρόνου εκπαίδευσης, για να

αποτρέψει το overfitting και ένα Dense layers (πλήρως διασυνδεδεμένο layer) για σκοπούς ταξινόμησης. Χρησιμοποιούμε μια συνάρτηση ενεργοποίησης softmax πριν από την τελική έξοδο. Το τελικό νευρωνικό έχει ως εξής:



Διάγραμμα 8 Το νευρωνικό δίκτυο του LSTM

Σε κώδικα το παραπάνω μεταφράζεται ως εξής:

```
# Model Training
vocab_size = 5000
embedding_size = 32
epochs=20
learning_rate = 0.1
decay_rate = learning_rate / epochs
momentum = 0.8

sgd = gradient_descent_v2.SGD(lr=learning_rate, momentum=momentum,
decay=decay_rate, nesterov=False)
# Build model
model= Sequential()
model.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model.add(Conv1D(filters=32, kernel_size=3, padding='same',
activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Bidirectional(LSTM(32)))
model.add(Dropout(0.4))
model.add(Dense(1, activation='softmax'))
print(model.summary())

# Compile model
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy', Precision(), Recall()])

# Train model
batch_size = 64
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    batch_size=batch_size, epochs=epochs, verbose=1)
```

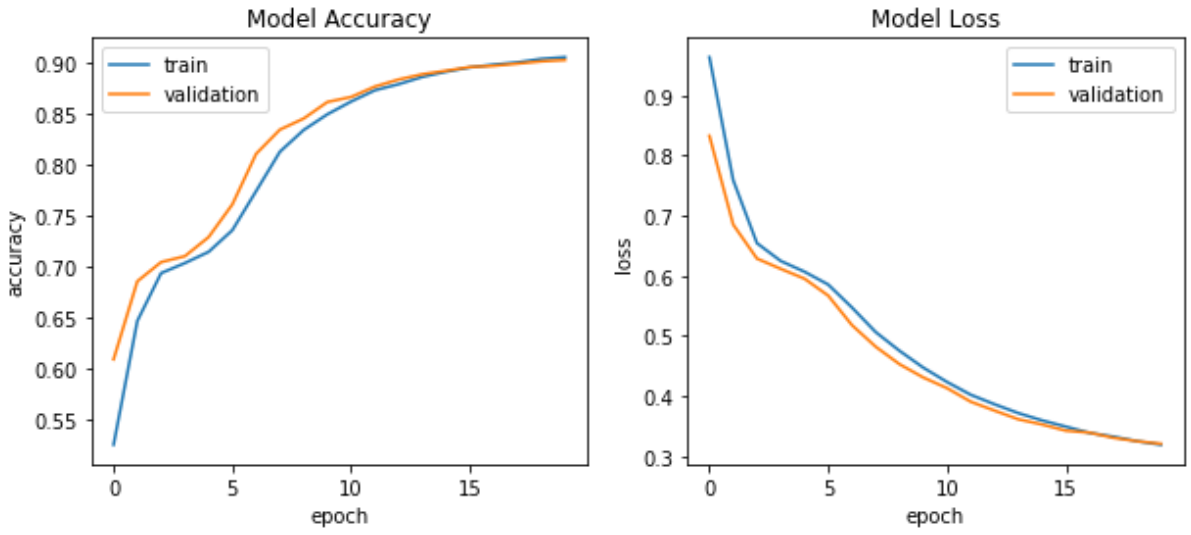
Στην συνέχεια, για να τυπώσουμε τες μετρικές που μας χρειάζονται για να κρίνουμε το μοντέλο μας, χρησιμοποιούμε τον παρακάτω κώδικα:

```
# Evaluate model on the test set
loss, accuracy, precision, recall = model.evaluate(X_test, y_test,
verbose=0)
# Print metrics
print('')
print('Accuracy  : {:.4f}'.format(accuracy))
print('Precision : {:.4f}'.format(precision))
print('Recall    : {:.4f}'.format(recall))
print('F1 Score  : {:.4f}'.format(f1_score(precision, recall)))
```

Το οποίο ως αποτέλεσμα έχει:

```
Accuracy  : 0.8952
Precision : 0.9004
Recall    : 0.8894
F1 Score  : 0.8949
```

Τέλος έχουμε και το ακόλουθο διάγραμμα που δείχνει το training history του LSTM:



Διάγραμμα 9 Το ιστορικό εκπαίδευσης του LSTM

9. Twitter Connection

Δημιουργούμε μια συνάρτηση που μας επιτρέπει να ενωθούμε με το twitter και να εξάγουμε tweets, βάση ενός δοθέν hashtag και μίας ημερομηνίας. Αρχικά, πρέπει να φτιάξουμε ένα λογαριασμό στο twitter και να συμπληρώσουμε συγκεκριμένες στοιχεία έτσι ώστε να μπορούμε να ενωθούμε σαν developers στο twitter. Αφού μας δώσουν τα credentials μας, μπορούμε να κάνουμε αναζήτηση στο Twitter για πρόσφατα tweets. Θα χρησιμοποιήσουμε ένα Cursor για να δημιουργήσουμε ένα αντικείμενο που θα κρατάει τα tweets που ανήκουν στο δοθέν hashtag. Για να δημιουργήσουμε ένα οποιοδήποτε ερώτημα, πρέπει να ορίσουμε: τον όρος αναζήτησης (το hashtag) και την ημερομηνία έναρξης της αναζήτησης. Αφού γίνει η αναζήτηση, περνάμε από όλα τα tweets που αποθηκεύτηκαν στον Cursor και τα κάνουμε append σε μία λίστα. Τέλος, την λίστα αυτή την αποθηκεύουμε σε ένα αρχείο csv.

```
# function to perform data extraction
def scrape(words, date_since, numtweet):
    # Enter the credentials obtained
    # from developer account
    consumer_key = 
    consumer_secret = 
    access_key = 
    access_secret = 


    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)

    # Creating DataFrame using pandas
    db = pd.DataFrame(columns=['username',
                               'description',
                               'location',
                               'following',
                               'followers',
                               'totaltweets',
                               'retweetcount',
                               'text',
                               'hashtags'])

    # We are using .Cursor() to search
    # through twitter for the required tweets.
    # The number of tweets can be
    # restricted using .items(number of tweets)
    tweets = tweepy.Cursor(api.search_tweets,
                           words, lang="en",
                           since_id=date_since,
                           tweet_mode='extended').items(numtweet)
```

```
    # .Cursor() returns an iterable object. Each item in
    # the iterator has various attributes
    # that we access to
    # get information about each tweet
    list_tweets = [tweet for tweet in tweets]

    # Counter to maintain Tweet Count
    i = 1

    # we will iterate over each tweet in the
    # list for extracting information about each tweet
    for tweet in list_tweets:
        username = tweet.user.screen_name
        description = tweet.user.description
        location = tweet.user.location
        following = tweet.user.friends_count
        followers = tweet.user.followers_count
        totaltweets = tweet.user.statuses_count
        retweetcount = tweet.retweet_count
        hashtags = tweet.entities['hashtags']

        # Retweets can be distinguished by
        # a retweeted_status attribute,
        # in case it is an invalid reference,
        # except block will be executed
        try:
            text = tweet.retweeted_status.full_text
        except AttributeError:
            text = tweet.full_text
            hashtext = list()
            for j in range(0, len(hashtags)):
                hashtext.append(hashtags[j]['text'])

                # Here we are appending all the
                # extracted information in the DataFrame
            ith_tweet = [username, description,
                         location, following,
                         followers, totaltweets,
                         retweetcount, text, hashtext]
            db.loc[len(db)] = ith_tweet

    filename = 'scraped_tweets.csv'

    # we will save our database as a CSV file.
    db.to_csv(filename)
```

10. Data Testing and Prediction

Στην τελική φάση, αρχικά ορίζουμε την παρακάτω συνάρτηση για το LSTM:

```
def predict_class(score):
    if score > 0.66:
        return "Positive"
    elif 0.33 <= score <= 0.66:
        return "Neutral"
    else:
        return "Positive"
```

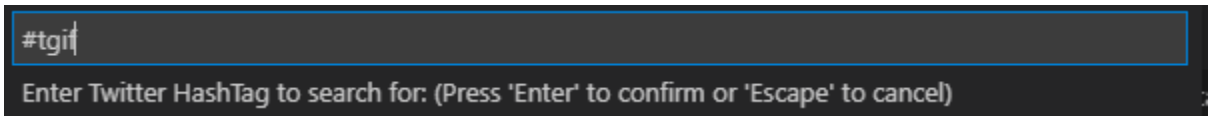
Το LSTM θα μας επιστρέψει ένα αριθμό. Καθώς, το σύνολο δεδομένων, δεν περιείχε δεδομένα που χαρακτηρίζονται ως Neutral, πρέπει να τα προσομοιώσουμε εμείς. Επομένως, ορίζουμε ξεχωριστά thresholds για το κάθε συναίσθημα. Δηλαδή, αν το LSTM επιστρέφει score μεγαλύτερο του 0.66, τότε το συναίσθημα είναι θετικό. Αν το score είναι μεταξύ 0.33 και 0.66 συμπεριλαμβανομένων, τότε το συναίσθημα είναι ουδέτερο. Σε κάθε άλλη περίπτωση το συναίσθημα είναι αρνητικό.

Ως main ορίζουμε την παρακάτω συνάρτηση:

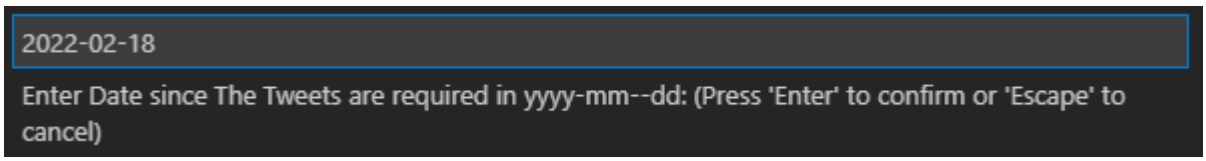
```
# Enter Hashtag and initial date
    words = input("Enter Twitter HashTag to search for: ")
    date_since = input("Enter Date since The Tweets are required in yyyy-mm-
-dd: ")

    # number of tweets you want to extract in one run
    numtweet = 100
    scrape(words, date_since, numtweet)
    print('Scraping has completed!')

    vectoriserFilename ="Vectoriser"
    lrFilename = "LogisticRegression"
    svcFilename = "LinearSVC"
    nbFilename = "MultinomialNB"
    xgbFilename = "XGBClassifier"
    dtcFilename = "DTClassifier"
    lstmFilename = 'LSTM'

    # Loading the models
    vectoriser, xgb_clf = load_model(vectoriserFilename, xgbFilename)
    vectoriser, nb_clf = load_model(vectoriserFilename, nbFilename)
    vectoriser, SVCmodel = load_model(vectoriserFilename, svcFilename)
    vectoriser, LRmodel = load_model(vectoriserFilename, lrFilename)
    vectoriser, dtc_clf = load_model(vectoriserFilename, dtcFilename)

    print(f'\n Tweets Sentiment Prediction')
```

```
df = pd.read_csv('scraped_tweets.csv')

text = df['text'].to_list()

# Prediction based on each model
print("------------ XGBoost ------------")
pr1 = predict(vectoriser, xgb_clf, text)
print(pr1.head())

print("------------ Multinomial Naive Bayes ------------")
pr2 = predict(vectoriser, nb_clf, text)
print(pr2.head())

print("------------ Linear SVC ------------")
pr3 = predict(vectoriser, SVCmodel, text)
print(pr3.head())

print("------------ Linear Regression ------------")
pr4 = predict(vectoriser, LRmodel, text)
print(pr4.head())

print("------------ Decision Tree ------------")
pr5 = predict(vectoriser, dtc_clf, text)
print(pr5.head())

print("------------ LSTM ------------")
temp = []
for i in text:
    temp.append((i, predict_class(i)))
pr6 = pd.DataFrame(temp, columns = ['text','sentiment'])
print(pr6.head())
```

Όπου αρχικά ζητάμε από τον χρήστη ένα hashtag και μια ημερομηνία από την οποία θα ξεκινήσουμε την αναζήτηση μας.



Στην συνέχεια δίνουμε μια ημερομηνία από την οποία ξεκινά η αναζήτηση για τα tweets που έχουν το hashtag που δόθηκε.



Τέλος, τα μοντέλα μας βγάζουν τα πιο κάτω αποτελέσματα με τις αντίστοιχες προβλέψεις:

```
------------------------- XGBoost --------------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...  Positive
1  Have a fabulous Saturday evening with that spe...  Positive
2  @TGIFridays Was this a mistake or you got a ne...  Positive
3  Be A Self-Starter for Incredible Passive Incom...  Positive
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...  Positive

----------------- Multinomial Naive Bayes ------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...  Positive
1  Have a fabulous Saturday evening with that spe...  Positive
2  @TGIFridays Was this a mistake or you got a ne...  Positive
3  Be A Self-Starter for Incredible Passive Incom...  Positive
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...  Positive

------------------------ Linear SVC ------------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...  Positive
1  Have a fabulous Saturday evening with that spe...  Positive
2  @TGIFridays Was this a mistake or you got a ne...  Positive
3  Be A Self-Starter for Incredible Passive Incom...  Positive
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...  Positive

---------------------- Linear Regression ---------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...  Positive
1  Have a fabulous Saturday evening with that spe...  Positive
2  @TGIFridays Was this a mistake or you got a ne...  Positive
3  Be A Self-Starter for Incredible Passive Incom...  Positive
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...  Positive

----------------------- Decision Tree ------------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...  Negative
1  Have a fabulous Saturday evening with that spe...  Negative
2  @TGIFridays Was this a mistake or you got a ne...  Positive
3  Be A Self-Starter for Incredible Passive Incom...  Negative
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...  Negative

--------------------------- LSTM ---------------------------
                                                text sentiment
0  What do you call a mollusk that's just short o...   Neutral
1  Have a fabulous Saturday evening with that spe...   Neutral
2  @TGIFridays Was this a mistake or you got a ne...   Neutral
3  Be A Self-Starter for Incredible Passive Incom...   Neutral
4  5 funds increasing $TGIF vs prior Q! #tgif SOF...   Neutral
```

Github Link

https://github.com/AndreasHadjigavriel/DT_Project