Τεχνητή Νοημοσύνη 2
Εργασία 1

Ιωάννης Καπετανγεώργης
111520180061

Περιεχόμενα

Μέρος Α – Απόδειξη Gradient MSE *3*

Μέρος Β - Classification *7*

Γενικά Σχόλια **7**

Μεθοδολογία **7**

Προεπεξεργασία Δεδομένων **8**

Επιλογή Vectorizer **9**

Μείωση των features 9

Softmax Classifier **10**

Αξιολόγηση του μοντέλου **11**

Κώδικας για την μελέτη **12**

Μελέτη για την εύρεση του κατάλληλου μοντέλου **14**

Εύρεση βέλτιστων παραμέτρων των Vectorizers 14

Εφαρμογή προεπεξεργασίας στα δεδομένα 15

Μοντέλο με το βέλτιστο score 16

Τροποποίηση του μοντέλου ώστε να μην παρουσιάζει overfit 17

Τελικό Μοντέλο **21**

Συμπεράσματα **22**

Προεπεξεργασία 22

Vectorization 22

Singular Value Decomposition (SVD) 23

Softmax 23

Γενικά 23

Μέρος Α – Απόδειξη Gradient MSE

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

Έχουμε ότι: $\mathrm{M}SE(w) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$

Από τον ορισμό του gradient έχουμε ότι το $\nabla_w MSE(w)$ είναι το διάνυσμα στήλη των μερικών παραγώγων. Δηλαδή:

$$\nabla_w MSE(w) = \begin{pmatrix} \frac{\partial}{\partial w_0} MSE(w) \\ \frac{\partial}{\partial w_1} MSE(w) \\ \vdots \\ \frac{\partial}{\partial w_n} MSE(w) \end{pmatrix}$$

Υπολογίζουμε την μερική παράγωγο ως προς w_0:

$$\begin{aligned} \frac{\partial}{\partial w_0} MSE(w) &= \frac{\partial}{\partial w_0} \frac{1}{m} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)}\right)^2 \\ &= \frac{1}{m} \frac{\partial}{\partial w_0} \sum_{i=1}^{m} \left(h_w(x^{(i)}) - y^{(i)}\right)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_0} \left(h_w(x^{(i)}) - y^{(i)}\right)^2 \end{aligned}$$

Για τον υπολογισμό του $\frac{\partial}{\partial w_0}\left(h_w(x^{(i)}) - y^{(i)}\right)^2$εφαρμόζουμε τον κανόνα της αλυσίδας και έχουμε:

$$\frac{\partial}{\partial w_0}\left(h_w(x^{(i)}) - y^{(i)}\right)^2 = 2\left(h_w(x^{(i)}) - y^{(i)}\right) \frac{\partial}{\partial w_0}\left(h_w(x^{(i)}) - y^{(i)}\right)$$

Για τον υπολογισμό του $\frac{\partial}{\partial w_0}\left(h_w(x^{(i)}) - y^{(i)}\right)$ έχουμε ότι:

$$\frac{\partial}{\partial w_0}\left(h_w(x^{(i)}) - y^{(i)}\right) = \frac{\partial}{\partial w_0} h_w(x^{(i)}) - \frac{\partial}{\partial w_0} y^{(i)}$$

Όπου:

$$\begin{aligned} \frac{\partial}{\partial w_0} h_w(x^{(i)}) &= \frac{\partial}{\partial w_0}\left(w \cdot x^{(i)}\right) = \frac{\partial}{\partial w_0}(w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) \\ &= \frac{\partial}{\partial w_0} w_0 x_0^{(i)} + \frac{\partial}{\partial w_0} w_1 x_1^{(i)} + \frac{\partial}{\partial w_0} w_2 x_2^{(i)} + \cdots + \frac{\partial}{\partial w_0} w_n x_n^{(i)} = x_0^{(i)} \end{aligned}$$

Και

$$\frac{\partial}{\partial w_0} y^{(i)} = 0$$

Δηλαδή: $\frac{\partial}{\partial w_0}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) = x_0^{(i)} \Longrightarrow$

$$\frac{\partial}{\partial w_0}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right)^2 = 2\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}$$

Άρα, τελικά:

$$\frac{\partial}{\partial w_0} MSE(w) = \frac{1}{m}\sum_{i=1}^{m} 2\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} = \frac{2}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)}$$

Αντίστοιχα αποδεικνύονται και οι μερικές παράγωγοι ως προς $w_1, w_2, \dots, w_n$
Επομένως έχουμε ότι:

$$\nabla_w MSE(w) = \begin{pmatrix} \frac{\partial}{\partial w_0} MSE(w) \\ \frac{\partial}{\partial w_1} MSE(w) \\ \vdots \\ \frac{\partial}{\partial w_n} MSE(w) \end{pmatrix} = \begin{pmatrix} \frac{2}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} \\ \frac{2}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_1^{(i)} \\ \vdots \\ \frac{2}{m}\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_n^{(i)} \end{pmatrix} = \frac{2}{m}\begin{pmatrix} \sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} \\ \sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_n^{(i)} \end{pmatrix}$$

Κάθε ένα από τα αθροίσματα μπορεί να γραφεί ως εξής:

$$\sum_{i=1}^{m}\left(h_w\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} = \sum_{i=1}^{m}\left(wx^{(i)} - y^{(i)}\right) x_0^{(i)}$$
$$= \left(wx^{(1)} - y^{(1)}\right) x_0^{(1)} + \left(wx^{(2)} - y^{(2)}\right) x_0^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right) x_0^{(m)}$$

Δηλαδή το $\nabla_w MSE(w)$ μπορεί να γραφεί ως εξής:

$$\nabla_w MSE(w) = \frac{2}{m}\begin{pmatrix} \left(wx^{(1)} - y^{(1)}\right) x_0^{(1)} + \left(wx^{(2)} - y^{(2)}\right) x_0^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right) x_0^{(m)} \\ \left(wx^{(1)} - y^{(1)}\right) x_1^{(1)} + \left(wx^{(2)} - y^{(2)}\right) x_1^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right) x_1^{(m)} \\ \vdots \\ \left(wx^{(1)} - y^{(1)}\right) x_n^{(1)} + \left(wx^{(2)} - y^{(2)}\right) x_n^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right) x_n^{(m)} \end{pmatrix}$$

Ορίζω έναν πίνακα **X** διαστάσεων $m \times (n+1)$

$$X = \begin{bmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Επίσης ορίζω το διάνυσμα **w** διάστασης $(n+1) \times 1$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

Έτσι, μπορεί να οριστεί το γινόμενο του πίνακα **X** με το διάνυσμα **W**.
Το αποτέλεσμα που προκύπτει είναι ένα κάθετο διάνυσμα διάστασης $m \times 1$.
Το γινόμενο που προκύπτει είναι το εξής:

$$Xw = \begin{bmatrix} w_0x_0^{(1)} + w_1x_1^{(1)} + \cdots + w_nx_n^{(1)} \\ w_0x_0^{(2)} + w_1x_1^{(2)} + \cdots + w_nx_n^{(2)} \\ \vdots \\ w_0x_0^{(m)} + w_1x_1^{(m)} + \cdots + w_nx_n^{(m)} \end{bmatrix} = \begin{bmatrix} wx^{(1)} \\ wx^{(2)} \\ \vdots \\ wx^{(m)} \end{bmatrix}$$

Τέλος ορίζουμε το διάνυσμα στήλη **y** : $y = [y^{(1)} \quad y^{(2)} \quad \dots \quad y^{(m)}]^T$
Το διάνυσμα **y** είναι επίσης διάστασης $m \times 1$.
Αφαιρώντας το διάνυσμα **y** από τον πίνακα $\boldsymbol{Xw}$ προκύπτει το:

$$Xw - y = \begin{bmatrix} wx^{(1)} - y^{(1)} \\ wx^{(2)} - y^{(2)} \\ \vdots \\ wx^{(m)} - y^{(m)} \end{bmatrix}$$

Επιστρέφοντας στον τύπο του $\nabla_w MSE(w)$ που έχουμε καταλήξει παραπάνω, κάθε ένα από τα αθροίσματα μπορεί να γραφεί ως εξής:

$$\nabla_w MSE(w) = \frac{2}{m}\begin{pmatrix} \left(wx^{(1)} - y^{(1)}\right)x_0^{(1)} + \left(wx^{(2)} - y^{(2)}\right)x_0^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right)x_0^{(m)} \\ \left(wx^{(1)} - y^{(1)}\right)x_1^{(1)} + \left(wx^{(2)} - y^{(2)}\right)x_1^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right)x_1^{(m)} \\ \vdots \\ \left(wx^{(1)} - y^{(1)}\right)x_n^{(1)} + \left(wx^{(2)} - y^{(2)}\right)x_n^{(2)} + \cdots + \left(wx^{(m)} - y^{(m)}\right)x_n^{(m)} \end{pmatrix}$$

$$= \frac{2}{m}\begin{pmatrix} [wx^{(1)} - y^{(1)} \quad wx^{(2)} - y^{(2)} \quad \dots \quad wx^{(m)} - y^{(m)}] \quad \left[x_0^{(1)} \quad x_0^{(2)} \quad \dots \quad x_0^{(m)}\right]^T \\ [wx^{(1)} - y^{(1)} \quad wx^{(2)} - y^{(2)} \quad \dots \quad wx^{(m)} - y^{(m)}] \quad \left[x_1^{(1)} \quad x_1^{(2)} \quad \dots \quad x_1^{(m)}\right]^T \\ \vdots \\ [wx^{(1)} - y^{(1)} \quad wx^{(2)} - y^{(2)} \quad \dots \quad wx^{(m)} - y^{(m)}] \quad \left[x_n^{(1)} \quad x_n^{(2)} \quad \dots \quad x_n^{(m)}\right]^T \end{pmatrix}$$

Παρατηρούμε ότι:

$$[wx^{(1)} - y^{(1)} \quad wx^{(2)} - y^{(2)} \quad \dots \quad wx^{(m)} - y^{(m)}] = (Xw - y)^T$$

(όπου **Xw-y** ο πίνακας που ορίσαμε παραπάνω)

Άρα

$$\nabla_w MSE(w) = \frac{2}{m}\begin{pmatrix} (Xw - y)^T \quad \left[x_0^{(1)} \quad x_0^{(2)} \quad \dots \quad x_0^{(m)}\right]^T \\ (Xw - y)^T \quad \left[x_1^{(1)} \quad x_1^{(2)} \quad \dots \quad x_1^{(m)}\right]^T \\ \vdots \\ (Xw - y)^T \quad \left[x_n^{(1)} \quad x_n^{(2)} \quad \dots \quad x_n^{(m)}\right]^T \end{pmatrix}$$

Τόσο το $(Xw-y)^T$ όσο και το $\begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}^T$ είναι διανύσματα. Πιο συγκεκριμένα είναι διανύσματα $1 \times m$ και $m \times 1$ διαστάσεων αντίστοιχα.

Άρα, ή πράξη πολλαπλασιασμού πινάκων $(Xw-y)^T \quad \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}^T$ αντιστοιχεί στο εσωτερικό γινόμενο των διανυσμάτων $(Xw-y)^T$ και $\begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}$

Ως γνωστών, στην πράξη του εσωτερικού γινομένου μπορούν να εναλλαγούν οι δύο όροι, Δηλαδή:

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a},$$

Λαμβάνοντας υπόψιν αυτό, αλλά και την εξής ιδιότητα:

. The dot product of two column vectors **a** and **b** can be computed as the single entry of the matrix product:

$$[\mathbf{a} \cdot \mathbf{b}] = \mathbf{a}^{\mathrm{T}}\mathbf{b},$$

which is written as $\mathbf{a}_i \mathbf{b}^i$ in Einstein summation convention.

Έχουμε:

$$\begin{aligned}(Xw-y)^T \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}^T &= (Xw-y) \cdot \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}^T \\ &= \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix}^T \cdot (Xw-y) \\ &= \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix} (Xw-y)\end{aligned}$$

Άρα ο τύπος του $\nabla_w MSE(w)$ μπορεί να γραφεί ως εξής:

$$\nabla_w MSE(w) = \frac{2}{m}\begin{pmatrix} \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix} & (Xw-y) \\ \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \end{bmatrix} & (Xw-y) \\ \vdots & \\ \begin{bmatrix} x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} & (Xw-y) \end{pmatrix}$$

Επίσης έχουμε ότι:

$$\begin{pmatrix} \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix} \\ \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} \end{pmatrix} = \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ \vdots & & & \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_0^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}^T = X^T$$

Όπου X ο πίνακας που έχουμε ορίσει.

Επομένως έχουμε:

$$\nabla_w MSE(w) = \frac{2}{m}\begin{pmatrix} \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & \dots & x_0^{(m)} \end{bmatrix} & (Xw-y) \\ \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \end{bmatrix} & (Xw-y) \\ \vdots & \\ \begin{bmatrix} x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix} & (Xw-y) \end{pmatrix} = \frac{2}{m}(X^T(Xw-y))$$

Δηλαδή αποδείξαμε το ζητούμενο:

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

Μέρος Β - Classification

Γενικά Σχόλια

- Στο παραδοτέο αρχείο (.ipynb) υπάρχει η υλοποίηση του sentiment classifier.
- Αρχικά, παραθέτω την τελική υλοποίηση με τον βέλτιστο classifier που κατάφερα να κατασκευάσω στο πρώτο (ενιαίο) cell του notebook. Το βέλτιστο αυτό μοντέλο περιγράφεται αναλυτικά στην παράγραφο Τελικό Μοντέλο.
- Στην συνέχεια, στο δεύτερο cell παραθέτω τον κώδικα μέσω του οποίου δοκίμασα διαφορετικά μοντέλα/τεχνικές έτσι ώστε να καταλήξω στον classifier του πρώτου κελιού. Το ποιες δοκιμές μπορούν να γίνουν μέσω αυτού του κώδικα καθώς και το πώς γίνονται περιγράφεται στην συνέχεια (στην παράγραφο Κώδικας για την μελέτη) καθώς και στο παραδοτέο αρχείο .ipynb στο αντίστοιχο text cell. Πιο συγκεκριμένα, μέσω των μεταβλητών που ορίζονται στην αρχή του cell μπορούμε να δοκιμάσουμε όλα όσα αναφέρονται στις παραγράφους: Προεπεξεργασία Δεδομένων, Επιλογή Vectorizer, Μείωση των features, Softmax Classifier με οποιονδήποτε συνδυασμό επιθυμούμε.
- **ΣΗΜΑΝΤΙΚΟ**: Για την δοκιμή του μοντέλου στο train set χρειάζεται να αλλάξει το path που ορίζεται από την εντολή:
  ```
  validation_set_location = r'vaccine_validation_set.csv'
  ```
  Η εντολή αυτή βρίσκεται στην αρχή του κάθε cell.
  Έτσι θα αντικατασταθεί το validation set με το train set και κατ' επέκταση ο classifier θα δοκιμαστεί και θα αξιολογηθεί για το test set.
- Στις παραγράφους που ακολουθούν θα περιγράψω αναλυτικά την διαδικασία/μελέτη που ακολούθησα έτσι ώστε να καταλήξω στην τελική μου υλοποίηση.

Μεθοδολογία

Η μεθοδολογία και τα βήματα που ακολούθησα για την παραγωγή του classifier είναι:

- Ανάγνωση των αρχείων train και validation και αποθήκευση των δεδομένων τους σε δυο dataframes (trainSet και validationSet αντίστοιχα).
- Προεπεξεργασία των δεδομένων τόσο του train set όσο και του validation set. "Καθαρίζουμε" τα δεδομένα έτσι ώστε κάθε tweet να περιέχει μόνο στοιχεία τα οποία μπορούν να προσφέρουν κάποια χρήσιμη πληροφορία.
- Επιλογή ενός vectorizer, αρχικοποίηση του με τις κατάλληλες παραμέτρους και δημιουργία της αντίστοιχης αναπαράστασης για κάθε ένα tweet.
  Εφαρμόζουμε fit_transform για το train set και transform για το validation set.
  Έτσι, κάθε ένα tweet αναπαρίσταται πλέον από ένα vector το οποίο περιέχει τιμές οι οποίες αντιστοιχούν σε features.
- Αρχικοποίηση του softmax classifier με τις κατάλληλες παραμέτρους, εκπαίδευση του με βάση το train set (fit) και δοκιμή στο validation set (predict).
- Αξιολόγηση του classification με βάση τις προβλέψεις που έκανε για το validation set.
  Για την αξιολόγηση χρησιμοποιείται: f1 score, precision και recall.
  Επίσης κατασκευάζονται και παρουσιάζονται τα learning curves έτσι ώστε να ανιχνευθεί αν το μοντέλο παρουσιάζει underfit ή overfit.

Προεπεξεργασία Δεδομένων

Μέσω της προεπεξεργασίας των δεδομένων κειμένου (στην περίπτωση μας tweets) καταφέρνουμε να απαλείψουμε λέξεις/στοιχεία τα οποία δεν προσφέρουν κάποια χρήσιμη πληροφορία.
Έτσι με την κατάλληλη προεπεξεργασία, προκύπτει μια καλύτερη αναπαράσταση από τον vectorizer η οποία δεν περιέχει άχρηστες πληροφορίες.
Πιο συγκεκριμένα οι τεχνικές που δοκίμασα είναι οι εξής:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions (αναφορά σε άλλους χρήστες του twitter), αφαίρεση των αριθμών και αφαίρεση των hashtags. Για τα παραπάνω χρησιμοποίησα τον tweet-preprocessor ο οποίος έχει κατασκευαστεί για αυτόν ακριβώς τον σκοπό.
- Αφαίρεση των emojis καθώς και αντικατάσταση τους από raw text (π.χ. 😀 → grinning_face)
- Αφαίρεση των συμβόλων, σημείων στίξης.
- Αφαίρεση των stop_words.
- Lemmatization μέσω του WordNetLemmatizer
- Stemming μέσω του SnowballStemmer

Κάποιες από αυτές τις τεχνικές ήταν αποδοτικές ενώ άλλες όχι. Επίσης, η αποδοτικότητα αυτών των τεχνικών συνδέεται και με τον vectorizer που θα επιλεχθεί καθώς και με τις παραμέτρους αυτού.
Τέλος αξίζει να σημειωθεί ότι ορισμένες τεχνικές (όπως το Lemmatization) ενώ είναι πολύ διαδεδομένες και κατά βάση πολύ αποδοτικές, αποδείχθηκε ότι με το δοθέν dataset δεν αποδίδουν και τόσο καλά.
Στην μελέτη/σύγκριση που ακολουθεί παρουσιάζεται αναλυτικά η απόδοση της κάθε τεχνικής καθώς και το ποιες από αυτές επέλεξα να εφαρμόσω τελικά.

Επιλογή Vectorizer

Όπως ήδη αναφέρθηκε, χρειαζόμαστε έναν vectorizer έτσι ώστε να δημιουργήσουμε μια αναπαράσταση για κάθε ένα tweet.
Οι vectorizers με τους οποίους πειραματίστηκα είναι:

- **Count Vectorizer**
- **TF-IDF Vectorizer**
- **Hashing Vectorizer**

Κάθε ένας από αυτούς χρησιμοποιεί κάποιες παραμέτρους οι οποίες ορίζονται κατα την αρχικοποίηση του.
Ανάλογα με την τιμή της κάθε παραμέτρου, ο vectorizer έχει διαφορετική συμπεριφορά και παράγει τα αντίστοιχα αποτελέσματα.
Οι παράμετροι με τις οποίες πειραματίστηκα είναι οι εξής:

- **min_df** και **max_df** για τους Count και TF-IDF Vectorizers. Μέσω των παραμέτρων αυτών ο vectorizer αγνοεί features τα οποία εμφανίζονται πολύ συχνά σε tweets και πολύ σπάνια αντίστοιχα. Ανάλογα με την τιμή των δύο παραμέτρων, ορίζεται η συχνότητα με βάση την οποία αγνοούμε ένα feature.
- **max_features** για τους Count και TF-IDF Vectorizers. Μέσω της παραμέτρου αυτής ορίζεται ο μέγιστος αριθμός των features που θα περιέχει κάθε αναπαράσταση. Πιο συγκεκριμένα, ο vectorizer κρατάει τα max_features με την μεγαλύτερη συχνότητα.
- **n_features** για τον Hashing Vectorizer. Μέσω της παραμέτρου αυτής ορίζεται ο αριθμός των features που θα περιέχει κάθε αναπαράσταση.
- **ngram_range** , μέσω της παραμέτρου αυτής ορίζεται ο αριθμός των λέξεων από τον οποίο μπορεί να αποτελείται ένα feature (π.χ. unigrams, bigrams).

Έτσι, δοκίμασα πολλούς συνδυασμούς παραμέτρων για κάθε μια μέθοδο για να εντοπίσω τον πιο αποδοτικό συνδυασμό.
Τα αποτελέσματα της μελέτης παρουσιάζονται στην συνέχεια.

Μείωση των features

Ως γνωστόν, για ένα μικρό dataset ένας μεγάλος αριθμός από features σε κάθε αναπαράσταση μπορεί να αποβεί μη αποδοτικός.
Όπως αναφέρθηκε παραπάνω ο αριθμός των features μπορεί να περιοριστεί μέσω των παραμέτρων των vectorizers.
Ωστόσο δοκίμασα και μια διαφορετική προσέγγιση για την μείωση των features μέσω decomposition.
Πιο συγκεκριμένα χρησιμοποίησα τον TruncatedSVD, ο οποίος δέχεται ένα sparse matrix (όπως αυτό προκύπτει από τους vectorizers) και το μετασχηματίζει μειώνοντας τον αριθμό των features της κάθε αναπαράστασης.
Ο αριθμός των τελικών features που θα προκύψουν ορίζεται από την παράμετρο n_components την τιμή της οποίας θα βρούμε μέσω της μελέτης που ακολουθεί.
Έτσι, δοκιμάζουμε να “χαλαρώσουμε” τις παραμέτρους των Vectorizers έτσι ώστε να παράγονται περισσότερα features και στην συνέχεια να τα μειώνουμε μέσω του SVD.

Softmax Classifier

Για τον softmax classifier χρησιμοποίησα την LogisticRegression της βιβλιοθήκης sklearn όριζοντας την παράμετρο `multi_class='multinomial'` έτσι ώστε να την προσαρμόσουμε στις ανάγκες μας για το multi-class πρόβλημα κατηγοριοποίησης που έχουμε.
Η LogisticRegression δέχεται πολλές παραμέτρους μέσω των οποίων προσαρμόζεται η συμπεριφορά της.
Οι παράμετροι με τις οποίες πειραματίστηκα είναι οι εξής:

- solver
- C
- max_iter

Για την επιλογή των κατάλληλων τιμών για τις παραμέτρους αυτές χρησιμοποίησα και την GridSearchCV, η οποία έπειτα από δοκιμές επιλέγει τον κατάλληλο συνδυασμό παραμέτρων.
Ώστοσο, τελικά επέλεξα διαφορετικές τιμές από αυτές που προέκυψαν από την GridSearchCV καθώς έπειτα από την μελέτη παρατήρησα καλύτερα αποτελέσματα.

Έπειτα από την επιλογή των κατάλληλων παραμέτρων για την αρχικοποίηση του μοντέλου, εκπαιδεύουμε το μοντέλο με τις αναπαραστάσεις που έχουν προκύψει από τον vectorizer για το train set και στην συνέχεια δοκιμάζουμε το μοντέλο κάνοντας predict για κάθε μια αναπαράσταση του validation set.

Αξιολόγηση του μοντέλου

Για την αξιολόγηση της κατηγοριοποίησης που προέκυψε από το μοντέλο χρησιμοποιώ τις εξής μετρικές:

- F1 score
- Precision
- Recall

Τις τιμές τις υπολογίζω χρησιμοποιώντας τις αντίστοιχες συναρτήσεις που παρέχονται από την βιβλιοθήκη sklearn.metrics.

Επίσης, κατασκευάζω τα learning curves του μοντέλου έτσι ώστε να ελέγξω την συμπεριφορά αυτού καθώς και το αν παρουσιάζει overfit ή underfit.
Για τον υπολογισμό των learning curves έχω κατασκευάσει μια συνάρτηση (με όνομα **find_learning_curves**) έτσι ώστε να χρησιμοποιούνται τα train set και validation set όπως αυτά μας έχουν δοθεί.

<u>Παρατήρηση</u>:

> Έτοιμες συναρτήσεις όπως η sklearn.model_selection.learning_curve εφαρμόζουν k-fold στο train set για την παραγωγή των δύο καμπυλών.
> Αντίθετα, εμείς θέλουμε η καμπύλη του validation να προκύπτει από το validation set το οποίο μας έχει δοθεί.

Έτσι, στην συνάρτηση που κατασκεύασα κάνω train το μοντέλο αυξάνοντας κάθε φορά το μέγεθος του υποσυνόλου του train set που χρησιμοποιείται για την εκμάθηση και κάνω κάθε φορά predict για ολόκληρο το validation set.
Για κάθε μια επανάληψη υπολογίζω το f1_score του prediction και το αποθηκεύω.
Στην τελευταία επανάληψη χρησιμοποιείται ολόκληρο το train set για την εκμάθηση και προφανώς ολόκληρο το validation set για τα predictions.
Τελικά παρουσιάζω τις καμπύλες χρησιμοποιώντας την βιβλιοθήκη pyplot.
Τα μεγέθη του train set για τα οποία εφαρμόζω την παραπάνω διαδικασία είναι:

[0.1*trainSize , 0.325*trainSize , 0.55*trainSize , 0.775*trainSize , trainSize]

όπου trainSize το μέγεθος του trainSet που μας δόθηκε.
Σαν επιπλέον γράφημα παρουσιάζω τα αντίστοιχα learning curves με την διαφορά ότι στον άξονα y αντί του f1_score παρουσιάζεται το MSE.

Για περαιτέρω μελέτη της συμπεριφοράς/απόδοσης του μοντέλου χρησιμοποιώ:

- Accuracy metric
- Κατασκευάζω το confusion matrix
- Χρησιμοποιώ την συνάρτηση classification_report της sklearn, η οποία παρουσιάζει τα precision, recall και f1-score για κάθε ένα label ξεχωριστά.

Κώδικας για την μελέτη

Στο τελευταίο κελί του παραδοτέου αρχείου (.ipynb) υπάρχει ο κώδικας που χρησιμοποιήθηκε έτσι ώστε να δοκιμάσω διάφορες τεχνικές, διαφορετικούς vectorizers, διάφορες παραμέτρους αρχικοποίησης τόσο των vectorizers όσο και του softmax και τελικά να συγκρίνω την απόδοση των δοκιμών αυτών.

Έτσι κατέληξα στο τελικό μοντέλο που παρουσιάζεται στην παράγραφο Τελικό Μοντέλο.

Πιο συγκεκριμένα μέσω του κώδικα:

- Μπορούν να δοκιμαστούν διαφορετικές τεχνικές προεπεξεργασίας δεδομένων καθώς και οποιοσδήποτε συνδυασμός αυτών.
- Στην συνέχεια δημιουργούνται τρεις διαφορετικές αναπαραστάσεις των δεδομένων μας χρησιμοποιώντας τρεις διαφορετικούς vectorizers: Count, TF-IDF και Hashing
- Κατασκευάζουμε τρία διαφορετικά μοντέλα softmax χρησιμοποιώντας τις τρεις αναπαραστάσεις από τους vectorizers
- Τέλος αξιολογούνται τα τρία αυτά μοντέλα μέσω accuracy, f1_score, precision, recall και learning curves.

Στην αρχή του κώδικα παρατηρούμε κάποιες μεταβλητές με κεφαλαία γράμματα. Μέσω αυτών καθορίζουμε το ποιες τεχνικές θέλουμε να εφαρμοστούν καθώς και τις παραμέτρους αρχικοποίησης των μεθόδων που χρησιμοποιούμε.

Για την προεπεξεργασία των δεδομένων μπορούμε να εφαρμόσουμε:

- Απαλοιφή των urls, mentions και των αριθμών (CLEAN_ULR_MENTIONS_NUMBERS)
- Απαλοιφή των stop words (CLEAN_STOPWORDS)
- Αντικατάσταση των emojis με το αντίστοιχο text (CLEAN_EMOJIS)
- Απαλοιφή συμβόλων και σημείων στίξης (CLEAN_PUNCUATIONS)
- Μετατροπή όλων των χαρακτήρων σε πεζούς (CONVERT_TO_LOWERCASE)
- Εφαρμογή Lemmatization (LEMMATIZATION)
- Εφαρμογή Stemming (STEMMING)

Αν η αντίστοιχη τιμή έχει τεθεί σε True τότε θα εφαρμοστεί η τεχνική αυτή ενώ αντίθετα αν η τιμή της μεταβλητής είναι False δεν θα εφαρμοστεί η αντίστοιχη τεχνική. Αν όλες οι παραπάνω μεταβλητές έχουν την τιμή False τότε δεν εφαρμόζεται καθόλου προεπεξεργασία στα δεδομένα μας.

Στην συνέχεια μπορούμε να ορίσουμε τις παραμέτρους που δέχονται οι vectorizers κατά την αρχικοποίηση τους και προσδιορίζουν την συμπεριφορά τους. Πιο συγκεκριμένα οι παράμετροι που μπορούμε να ορίσουμε είναι:

- min_df (Count και TF-IDF)
- max_df (Count και TF-IDF)
- max_features (Count και TF-IDF)
- n_features (Hashing Vectorizer)
- n_gram (και για τους τρεις vectorizers)

Έπειτα από την δημιουργία των τριών αναπαραστάσεων μέσω των τριών vectorizers μπορούμε να επιλέξουμε μέσω της μεταβλητής ENABLE_SVD το αν θα εφαρμοστεί decomposition για την μείωση του αριθμού των features μέσω του SVD θέτοντας την τιμή της σε True ή False αντίστοιχα. Αν θέλουμε να εφαρμόσουμε decomposition χρειάζεται να ορίσουμε και των τελικό αριθμό των features που θα παραχθούν μέσω της μεταβλητής N_COMPONENTS.

Τέλος μπορούμε να ορίσουμε τις μεταβλητές αρχικοποίησης του softmax. Πιο συγκεκριμένα ορίζουμε:

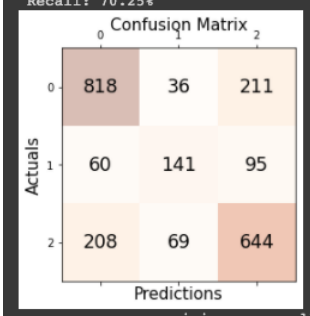
- solver (SOLVER)
- C (C_VALUE)
- max_iter (MAX_ITER)

Αφού ορίσουμε κατάλληλα τις παραπάνω μεταβλητές, εκτελούμε το cell, δημιουργούνται τα 3 διαφορετικά μοντέλα και τελικά αξιολογούνται βάση των μετρικών που αναφέρθηκαν.

Μελέτη για την εύρεση του κατάλληλου μοντέλου

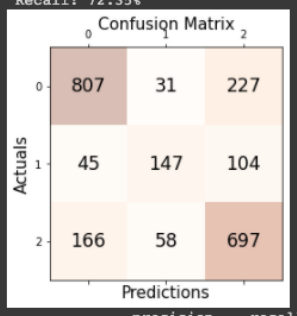
Αρχικά δοκιμάζουμε το μοντέλο χωρίς να εφαρμόσουμε καθόλου προεπεξεργασία στα δεδομένα και αφήνοντας τις default τιμές στους vectorizers.
Για τον softmax χρειάζεται να επιλέξουμε έναν **solver**, δοκιμάζοντας όλους τους δυνατούς την καλύτερη απόδοση είχε ο **newton-cg.** Επίσης για την παράμετρο **C** επέλεξα την τιμή **3** καθώς με αυτήν παρατήρησα καλύτερα αποτελέσματα. Τέλος, αυξάνουμε τον αριθμό των μέγιστων επαναλήψεων σε 1000 έτσι ώστε κάθε φορά να συγκλίνει η μέθοδος (δηλαδή **max_iter=1000**).
Κάνουμε τρεις διαδοχικές δοκιμές, μια για κάθε vectorizer (count, tf-df, hash).
Τα αποτελέσματα που προκύπτουν είναι τα εξής:

```
Softmax Regression using vectorizer: Count Vectorizer
 Accuracy: 70.25%
 f1 score: 70.03%
 Precision: 69.95%
 Recall: 70.25%
```



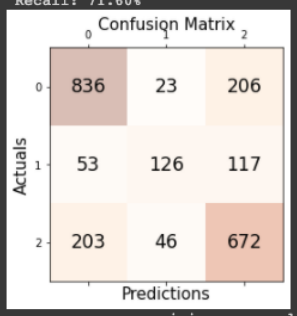
	precision	recall	f1-score	support
0	0.75	0.77	0.76	1065
1	0.57	0.48	0.52	296
2	0.68	0.70	0.69	921
accuracy			0.70	2282
macro avg	0.67	0.65	0.66	2282
weighted avg	0.70	0.70	0.70	2282

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 72.35%
 f1 score: 72.20%
 Precision: 72.44%
 Recall: 72.35%
```



	precision	recall	f1-score	support
0	0.79	0.76	0.77	1065
1	0.62	0.50	0.55	296
2	0.68	0.76	0.72	921
accuracy			0.72	2282
macro avg	0.70	0.67	0.68	2282
weighted avg	0.72	0.72	0.72	2282

```
Softmax Regression using vectorizer: Hashing Vectorizer
 Accuracy: 71.60%
 f1 score: 71.14%
 Precision: 71.37%
 Recall: 71.60%
```



	precision	recall	f1-score	support
0	0.77	0.78	0.78	1065
1	0.65	0.43	0.51	296
2	0.68	0.73	0.70	921
accuracy			0.72	2282
macro avg	0.70	0.65	0.66	2282
weighted avg	0.71	0.72	0.71	2282

Παρατηρούμε πολύ κοντινά αποτελέσματα και με τους τρεις vectorizers, με τον καλύτερο να είναι ο tf-idf.

Στην συνεχεια, δοκίμασα να εφαρμόσω προεπεξεργασία στα δεδομένα. Ωστόσο δεν κατάφερα να βελτιώσω καθόλου την απόδοση.
Έτσι, συνέχισα με την εύρεση των κατάλληλων παραμέτρων για τους vectorizers.

Εύρεση βέλτιστων παραμέτρων των Vectorizers

Έπειτα από την δοκιμή διαφόρων συνδυασμών κατέληξα στις εξής βέλτιστες παραμέτρους:

- min_df = 1 (default)
- max_df = 1.0 (default)
- max_features = 100.000
- n_gram = (1,3)
- n_features = 100.000

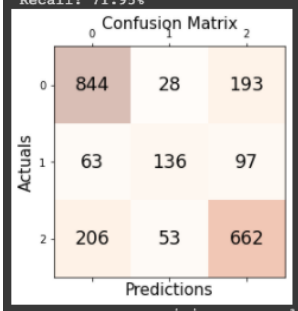
Δηλαδή μειώνουμε τον αριθμό των features που χρησιμοποιούνται μέσω της παραμέτρου max_features. Αντίστοιχη μείωση των features μπορούμε να πετύχουμε και μέσω των min_df και max_df. Ωστόσο, όποιον συνδυασμό και να δοκίμασα υστερούσε σε απόδοση σε σχέση με τον περιορισμό μέσω του max_features.
Επίσης, μέσω της παραμέτρου n_gram χρησιμοποιούμε unigram, bigram και trigram για τον καθορισμό των features.

Αρχικοποιώντας τους vectorizers με τις παραπάνω παραμέτρους εκτελούμε πάλι τρεις διαδοχικές δοκιμές, μια για κάθε vectorizer.

(Οι παράμετροι του softmax παραμένουν ίδιες, καθώς παρουσιάζουν την καλύτερη απόδοση δηλαδή: solver='newton-cg', C=3, max_iter=1000)

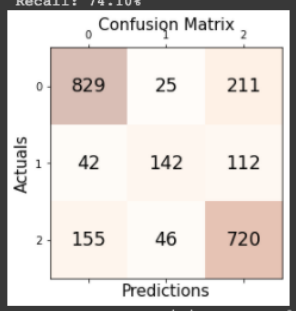
Τα αποτελέσματα που προκύπτουν είναι τα εξής:

```
Softmax Regression using vectorizer: Count Vectorizer
 Accuracy: 71.95%
 f1 score: 71.58%
 Precision: 71.58%
 Recall: 71.95%
```



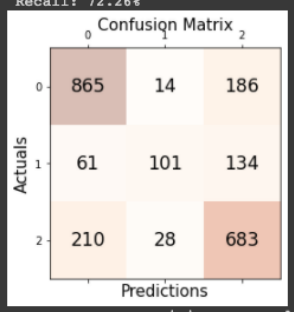
	precision	recall	f1-score	support
0	0.76	0.79	0.78	1065
1	0.63	0.46	0.53	296
2	0.70	0.72	0.71	921
accuracy			0.72	2282
macro avg	0.69	0.66	0.67	2282
weighted avg	0.72	0.72	0.72	2282

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 74.10%
 f1 score: 73.83%
 Precision: 74.22%
 Recall: 74.10%
```



	precision	recall	f1-score	support
0	0.81	0.78	0.79	1065
1	0.67	0.48	0.56	296
2	0.69	0.78	0.73	921
accuracy			0.74	2282
macro avg	0.72	0.68	0.69	2282
weighted avg	0.74	0.74	0.74	2282

```
Softmax Regression using vectorizer: Hashing Vectorizer
 Accuracy: 72.26%
 f1 score: 71.31%
 Precision: 72.18%
 Recall: 72.26%
```



	precision	recall	f1-score	support
0	0.76	0.81	0.79	1065
1	0.71	0.34	0.46	296
2	0.68	0.74	0.71	921
accuracy			0.72	2282
macro avg	0.72	0.63	0.65	2282
weighted avg	0.72	0.72	0.71	2282

Παρατηρούμε αύξηση της απόδοσης και για τους τρεις vectorizers. Καλύτερος παραμένει ο TF-IDF.

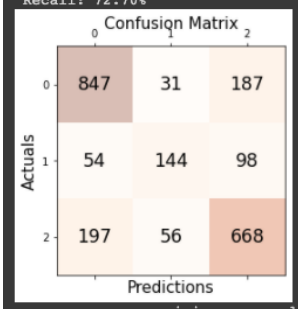
Εφαρμογή προεπεξεργασίας στα δεδομένα

Στην συνέχεια της μελέτης δοκίμασα ξανά να εφαρμόσω προεπεξεργασία στα δεδομένα.
Οι τεχνικές που δοκίμασα αναφέρονται στην παράγραφο [Προεπεξεργασία Δεδομένων](#).
Δοκίμασα κάθε συνδυασμό αυτών των τεχνικών.
Τελικά, σε αντίθεση με πριν βελτιώθηκε, έστω και ελάχιστα, η απόδοση.
Κατέληξα ότι την καλύτερη απόδοση πετυχαίνουμε εφαρμόζοντας τα εξής:

- Μετατροπή όλων των χαρακτήρων σε πεζούς (lower case).
- Αφαίρεση των links, των mentions και των αριθμών καθώς δεν προσφέρουν καμία πληροφορία σχετικά με το “νόημα” της πρότασης/
- Αντικατάσταση των emojis σε raw text (π.χ. 😀 → grinning_face) καθώς πολλές φορές αποτυπώνουν πληροφορία σχετικά με το συναίσθημα.
- Αφαίρεση των συμβόλων, σημείων στίξης με εξαίρεση τα hashtag καθώς αποτελούν λέξεις κλειδιά για ένα tweet. Επίσης, δεν αφαιρώ τα θαυμαστικά και τα ερωτηματικά καθώς παρατήρησα καλυτερη απόδοση, αφού και αυτά πολλές φορές δηλώνουν συναίσθημα (π.χ. εκνευρισμό).

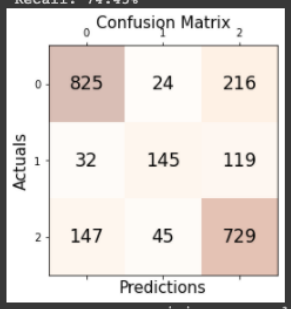
Πλέον τα αποτελέσματα που παίρνουμε για τους τρεις vectorizers είναι τα εξής:

```
Softmax Regression using vectorizer: Count Vectorizer
 Accuracy: 72.70%
 f1 score: 72.41%
 Precision: 72.38%
 Recall: 72.70%
```



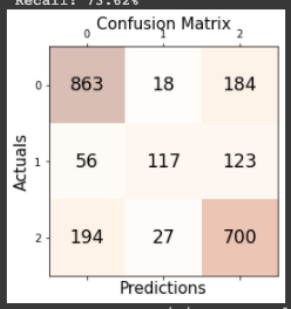
	precision	recall	f1-score	support
0	0.77	0.80	0.78	1065
1	0.62	0.49	0.55	296
2	0.70	0.73	0.71	921
accuracy			0.73	2282
macro avg	0.70	0.67	0.68	2282
weighted avg	0.72	0.73	0.72	2282

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 74.45%
 f1 score: 74.24%
 Precision: 74.79%
 Recall: 74.45%
```



	precision	recall	f1-score	support
0	0.82	0.77	0.80	1065
1	0.68	0.49	0.57	296
2	0.69	0.79	0.73	921
accuracy			0.74	2282
macro avg	0.73	0.69	0.70	2282
weighted avg	0.75	0.74	0.74	2282

```
Softmax Regression using vectorizer: Hashing Vectorizer
 Accuracy: 73.62%
 f1 score: 72.92%
 Precision: 73.61%
 Recall: 73.62%
```



	precision	recall	f1-score	support
0	0.78	0.81	0.79	1065
1	0.72	0.40	0.51	296
2	0.70	0.76	0.73	921
accuracy			0.74	2282
macro avg	0.73	0.66	0.68	2282
weighted avg	0.74	0.74	0.73	2282

Έχοντας βρει τις βέλτιστες παραμέτρους για τους vectorizers αλλά και την αποδοτικότερη προεπεξεργασία των δεδομένων, Δοκίμασα ξανά όλους τους συνδυασμούς για τις παραμέτρους του softmax.
Ώστοσο, την καλύτερη απόδοση συνέχιζαν να μου δίνουν οι ίδιες παράμετροι, δηλαδή:

solver='newton-cg', C=3, max_iter=1000

Μοντέλο με το βέλτιστο score

Έτσι συνοψίζοντας όλα τα παραπάνω καταλήγουμε πως την καλύτερη ακρίβεια για τις προβλέψεις στο validation set πετυχαίνουμε ως εξής:

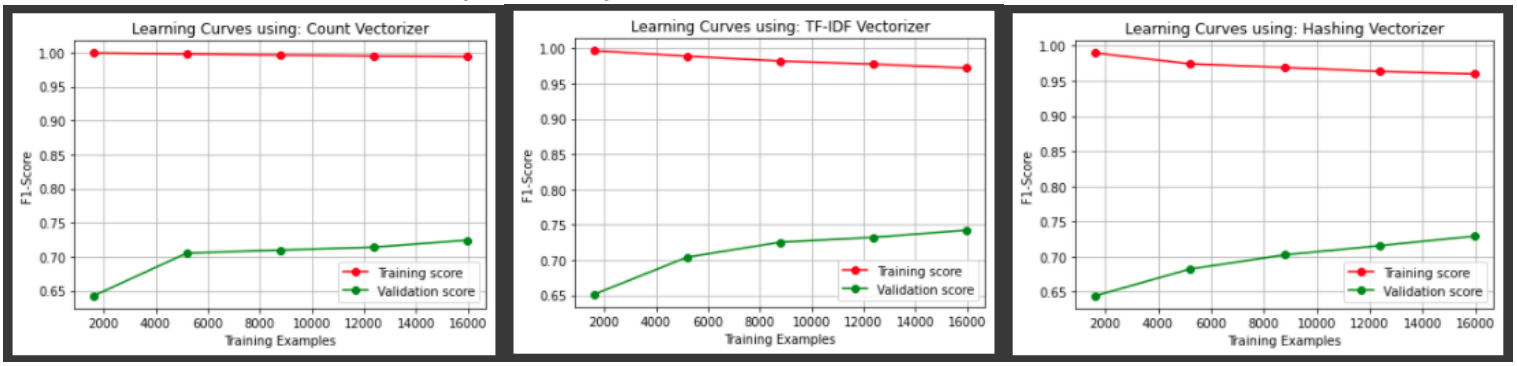
- Εφαρμόζοντας την εξής προεπεξεργασία στα δεδομένα:
  - Μετατροπή του κειμένου σε lower case
  - Αφαίρεση links, mentions και αριθμών
  - Αντικατάσταση των emojis με το αντίστοιχο text
  - Αφαίρεση συμβόλων και σημείων στίξης
- Χρησιμοποιώντας τον TF-IDF Vectorizer, με τις εξής παραμέτρους:
  - min_df = 1 (default)
  - max_df = 1.0 (default)
  - max_features = 100.000
  - n_gram = (1,3)
- Τέλος, χρησιμοποιώντας τον softmax με τις εξής παραμέτρους:
  - solver = newton-cg
  - C = 3
  - max_iter = 1000

Τα score που πετυχαίνουμε για το δοθέν validation set είναι:

- **Accuracy : 74.45%**
- **f1 score : 74.24%**
- **Precision : 74.79%**
- **Recall: 74.45%**

Επίσης, από τα επιμέρους f1, precision και recall για κάθε label παρατηρούμε πως το μοντέλο μας υστερεί περισσότερο στην αναγνώριση των tweets που ανήκουν στο label 1, δηλαδή είναι ουδέτερα.
Ωστόσο, μελετώντας ένα προς ένα τα tweets τα οποία το μοντέλο μας δεν καταφέρνει να κατηγοριοποιήσει και κάνοντας επιπλέον δοκιμές δεν κατάφερα να αυξήσω την απόδοση.

Αφού καταλήξαμε στο μοντέλο με την καλύτερη απόδοση ήρθε η ώρα να το μελετήσουμε μέσω των learning curves.
Στην συνέχεια, παρουσιάζω τα learning curves των μοντέλων από τις τελευταίες τρεις διαδοχικές εκτελέσεις (μια για κάθε vectorizer) με τις παραμέτρους που περιγράφηκαν παραπάνω, δηλαδή τις εκτελέσεις με τα βέλτιστα scores.



Από τα παραπάνω learning curves εύκολα παρατηρούμε πως το μοντέλο μας (ανεξαρτήτως vectorizer) παρουσιάζει overfit.

<u>ΠΑΡΑΤΗΡΗΣΗ</u>: αντίστοιχα learning curves προέκυπταν και για όλες τις δοκιμές που ανέφερα παραπάνω, για αυτό και δεν παρουσιάστηκαν. Δηλαδή σε κάθε περίπτωση το μοντέλο μας έκανε overfit. Έτσι, παρουσιάζω τα learning curves μονο για το μοντέλο με την καλύτερη απόδοση και συνεχίζω την μελέτη/βελτίωση πάνω σε αυτό.

Τροποποίηση του μοντέλου ώστε να μην παρουσιάζει overfit

Οι βασικοί λόγοι για την ύπαρξη του overfit είναι:

- Το train set δεν είναι “καθαρό” και περιέχει θόρυβο
- Το μέγεθος του train set δεν είναι αρκετά μεγάλο
- Το μοντέλο που χρησιμοποιούμε είναι αρκετά πολύπλοκο σε σχέση με το “πρόβλημα” που προσπαθούμε να λύσουμε
- Το μοντέλο παρουσιάζει υψηλό variance

Στην περίπτωση μας έχουμε ήδη εφαρμόσει προεπεξεργασία στα δεδομένα και τα έχουμε “καθαρίσει”. Επίσης όπως έχει ήδη αναφερθεί δοκίμασα να τα καθαρίσω ακόμη περισσότερο με επιπλέον τεχνικές, ωστόσο σε κάθε περίπτωση το μοντέλο παρουσίαζε overfit.
Όσον αφορά το μέγεθος του train set, προφανώς δεν μπορούμε να κάνουμε κάτι καθώς εργαζόμαστε με τα συγκεκριμένα train και validation set που μας έχουν δοθεί.

Έτσι, το μόνο που μας μένει είναι να απλοποιήσουμε το μοντέλο μας.

Έχοντας δοκιμάσει διαφορετικούς vectorizers (πιο απλούς όπως ο Count και πιο σύνθετους όπως ο TF-IDF) αλλά και πολλούς συνδυασμούς παραμέτρων τόσο στους vectorizers όσο και στον softmax παρατηρίσαμε παρόμοια αποτελέσματα.
Η μόνη λύση που απομένει για να απλοποιήσουμε το μοντέλο είναι να δοκιμάσουμε να μειώσουμε τον αριθμό των features που προκύπτουν από τους vectorizers.
Έτσι θα προκύπτει μια απλούστερη αναπαράσταση για κάθε ένα tweet, λαμβάνοντας υπόψη τα σημαντικότερα χαρακτηριστικά του και αγνοώντας τα λιγότερο συχνά/σημαντικά.

Μείωση των features μέσω των παραμέτρων των vectorizers

Την μείωση των features θα επιχειρήσω να επιτύχω μέσω των παραμέτρων min_df και max_df (ή της παραμέτρου max_features) για τους Count και TF-IDF vectorizers και μέσω της παραμέτρου n_features για τον Hashing vectorizer.
Έτσι η μελέτη συνεχίζεται κάνοντας δοκιμές μικραίνοντας σταδιακά τον αριθμό των features έως ότου να μην παρουσιάζει overfit το μοντέλο μας.
Όσον αφορά τον Count και TF-IDF vectorizer παρατήρησα καλύτερα αποτελέσματα μειώνοντας τα features μέσω της παραμέτρου max_features παρά μέσω των min_df και max_df. Πιο συγκεκριμένα, όποιον συνδυασμό min_df και max_df δοκίμασα σε κάθε περίπτωση η ακρίβεια ήταν χειρότερη. Έτσι, εργάζομαι τροποποιόντας την παράμετρο max_features.

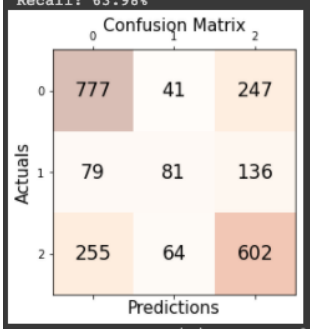
Τελικά, έπειτα από δοκιμές με σταδιακή μείωση του αριθμού των features κατέληξα πως η βέλτιστη τιμή είναι 600. Δηλαδή:

- max_features = 600 (Για Count και TF-IDF vectorizers)
- n_features = 600 (Για Hashing vectorizer)

Όπως είναι αναμενόμενο, τα scores του μοντέλου για το validation set μειώνονται. Ωστόσο πλέον το μοντέλο μας δεν κάνει overfit και κατ' επέκταση θα είναι αποδοτικότερο σε ένα "άγνωστο" test set.

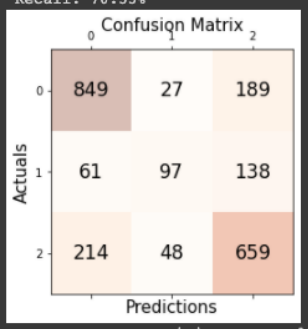
Τα αποτελέσματα έπειτα από την μείωση των features σε 600 είναι τα εξής:

```
Softmax Regression using vectorizer: Hashing Vectorizer
 Accuracy: 63.98%
 f1 score: 63.18%
 Precision: 62.95%
 Recall: 63.98%
```



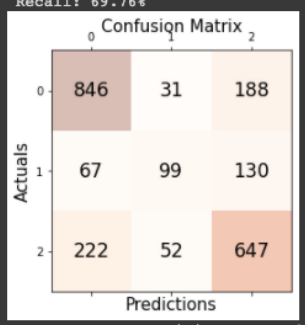
	precision	recall	f1-score	support
0	0.70	0.73	0.71	1065
1	0.44	0.27	0.34	296
2	0.61	0.65	0.63	921
accuracy			0.64	2282
macro avg	0.58	0.55	0.56	2282
weighted avg	0.63	0.64	0.63	2282

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 70.33%
 f1 score: 69.47%
 Precision: 69.54%
 Recall: 70.33%
```



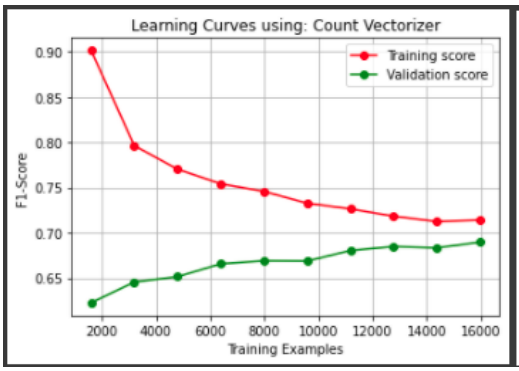
	precision	recall	f1-score	support
0	0.76	0.80	0.78	1065
1	0.56	0.33	0.41	296
2	0.67	0.72	0.69	921
accuracy			0.70	2282
macro avg	0.66	0.61	0.63	2282
weighted avg	0.70	0.70	0.69	2282

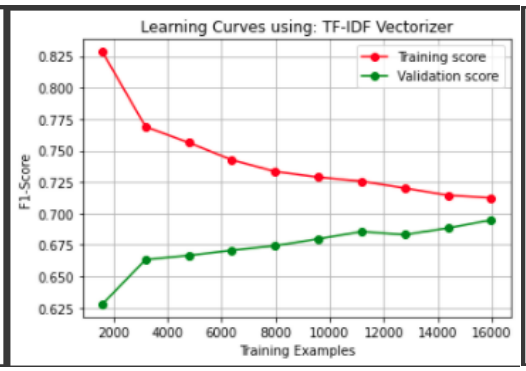
```
Softmax Regression using vectorizer: Count Vectorizer
 Accuracy: 69.76%
 f1 score: 68.96%
 Precision: 68.90%
 Recall: 69.76%
```

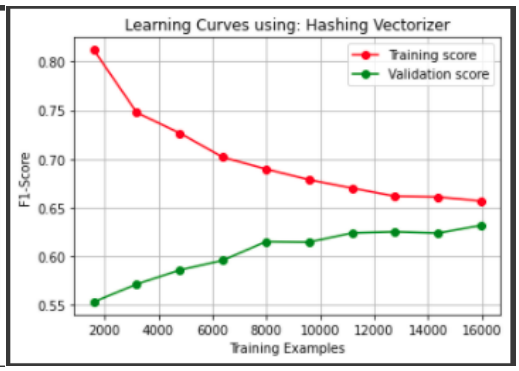


	precision	recall	f1-score	support
0	0.75	0.79	0.77	1065
1	0.54	0.33	0.41	296
2	0.67	0.70	0.69	921
accuracy			0.70	2282
macro avg	0.65	0.61	0.62	2282
weighted avg	0.69	0.70	0.69	2282

Τα αντίστοιχα learning curves είναι:







Παρατηρούμε πως πλέον όσο αυξάνεται το μέγεθος του train set οι δυο καμπύλες έρχονται όλο και πιο κοντά. Τελικά, χρησιμοποιώντας ολόκληρο το train set για την εκμάθηση οι δύο καμπύλες έχουν πολύ μικρό gap.
Πιο συγκεκριμένα, για τον TF-IDF οι καμπύλες τελικά απέχουν κατά λιγότερο από 0.025.
Επίσης παρατηρούμε καλύτερη απόδοση χρησιμοποιώντας Count και TF-IDF vectorizers, με τον TF-IDF να πετυχαίνει τελικά λίγο καλύτερο score στο validation set.

Πλέον το μοντέλο μας δεν παρουσιάζει overfit.

Μείωση των features μέσω decomposition - SVD

Στην συνέχεια της μελέτης θα δοκιμάσω να μειώσω τα features που χρησιμοποιούνται για κάθε αναπαράσταση μέσω του singular value decomposition (SVD), αντί να το κάνω μέσω των παραμέτρων των vectorizers.
Πιο συγκεκριμένα θα χρησιμοποιήσω την sklearn.decomposition.TruncatedSVD η οποία δέχεται σαν input sparse matrix, ακριβώς όπως αυτά προκύπτουν από τους vectorizers.

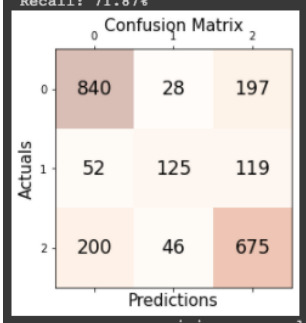
Αρχικά θα εφαρμόσω το decomposition στο βέλτιστο μοντέλο που παρουσιαστηκε στην παράγραφο Μοντέλο με το βέλτιστο score.

Κρατώντας τις παραμέτρους των vectorizers και του softmax ως έχουν, εφαρμόζουμε decomposition μειώνοντας σταδιακά τον αριθμό των τελικών features που προκύπτουν από αυτό.
Τελικά παρατηρούμε την καλύτερη απόδοση μειώνοντας τον αριθμό των features στα 600, δηλαδή ορίζουμε την παράμετρο του TruncatedSVD : **n_components=600**

Παρατήρηση: Λόγω της τυχαιότητας που χρησιμοποιεί ο SVD, τα αποτελέσματα από εκτέλεση σε εκτέλεση μπορεί να διαφέρουν ελάχιστα. Σε κάθε περίπτωση όμως παρουσιάζουν την “καλή” συμπεριφορά που περιγράφεται στην συνέχεια

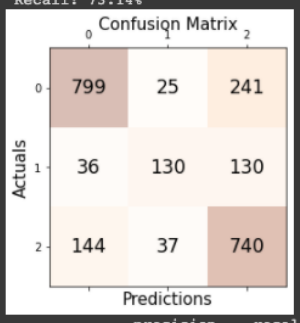
Τα αποτελέσματα που προκύπτουν είναι τα εξής:

```
Softmax Regression using vectorizer: Count Vectorizer
 Accuracy: 71.87%
 f1 score: 71.40%
 Precision: 71.54%
 Recall: 71.87%
```



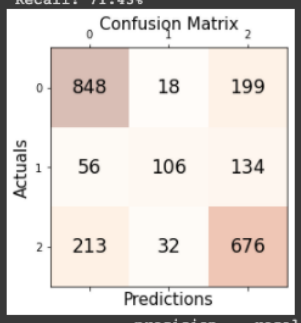
	precision	recall	f1-score	support
0	0.77	0.79	0.78	1065
1	0.63	0.42	0.51	296
2	0.68	0.73	0.71	921
accuracy			0.72	2282
macro avg	0.69	0.65	0.66	2282
weighted avg	0.72	0.72	0.71	2282

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 73.14%
 f1 score: 72.79%
 Precision: 73.75%
 Recall: 73.14%
```



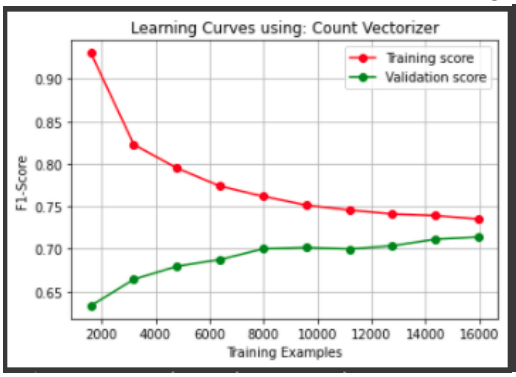
	precision	recall	f1-score	support
0	0.82	0.75	0.78	1065
1	0.68	0.44	0.53	296
2	0.67	0.80	0.73	921
accuracy			0.73	2282
macro avg	0.72	0.66	0.68	2282
weighted avg	0.74	0.73	0.73	2282

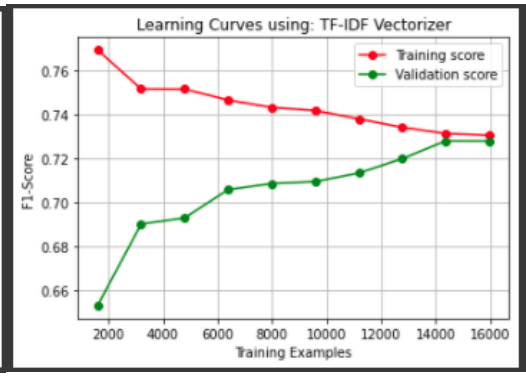
```
Softmax Regression using vectorizer: Hashing Vectorizer
 Accuracy: 71.43%
 f1 score: 70.63%
 Precision: 71.28%
 Recall: 71.43%
```

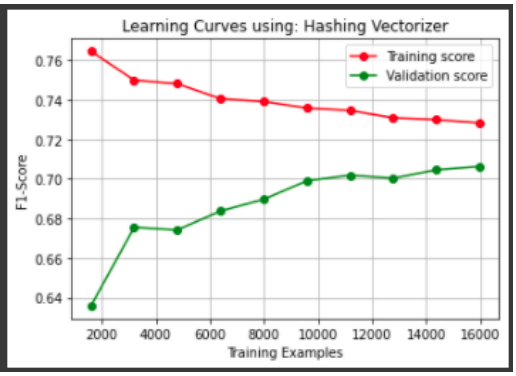


	precision	recall	f1-score	support
0	0.76	0.80	0.78	1065
1	0.68	0.36	0.47	296
2	0.67	0.73	0.70	921
accuracy			0.71	2282
macro avg	0.70	0.63	0.65	2282
weighted avg	0.71	0.71	0.71	2282

και τα αντίστοιχα learning curves είναι:







Συγκρίνοντας τα αυτά αποτελέσματα με τα αντίστοιχα που προέκυψαν από την μείωση των features μέσω των παραμέτρων των vectorizers παρατηρούμε:

- Έχουν αυξηθεί τα f1_score, precision και recall και για τους τρεις vectorizers
- Τα learning curves πλησιάζουν ακόμα περισσότερο. Συγκεκριμένα για τον TF-IDF οι δυο καμπύλες σχεδόν ταυτίζονται.
- Έχουν μειωθεί τα scores που επιτυγχάνονται για το training set. Συγκεκριμένα, παρατηρούμε ότι για τον TF-IDF το training score ακόμα και για την δοκιμή με το μικρότερο υποσύνολο του training set είναι αρκετά χαμηλό. Ταυτόχρονα όμως, έχει αυξηθεί το validation score.

Έτσι καταλήγουμε στο ότι η μείωση των features μέσω του SVD είναι αποδοτικότερη, καθώς πλέον το μοντέλο μας δεν παρουσιάζει καθόλου overfit ενώ ταυτόχρονα έχει αυξηθεί και η απόδοση του στο validation set.

Ωστόσο πρέπει να σημειωθεί ότι η χρήση του SVD προσθέτει λίγο επιπλέον χρόνο για την κατασκευή του μοντέλου. Όμως λαμβάνοντας υπόψη τον λίγο χρόνο που χρειάζεται το μοντέλο για την εκμάθηση (λόγω του σχετικά μικρού train set) καθώς και την αύξηση της απόδοσης που μας προσφέρει ο SVD, πιστεύω πως αξίζει να τον χρησιμοποιήσουμε.

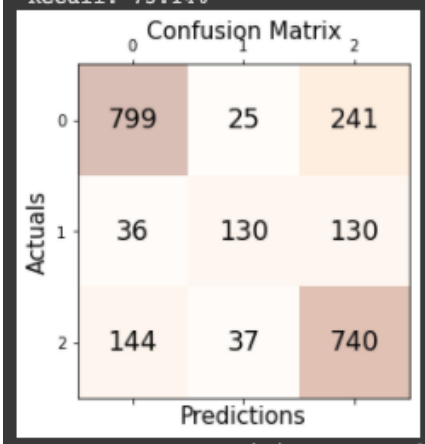
Τελικό Μοντέλο

Λαμβάνοντας υπόψη όλα όσα αναφέρθηκαν στην παράγραφο [Μελέτη για την εύρεση του κατάλληλου μοντέλου]() καταλήγουμε στο τελικό βέλτιστο μοντέλο το οποίο προκύπτει ως εξής:

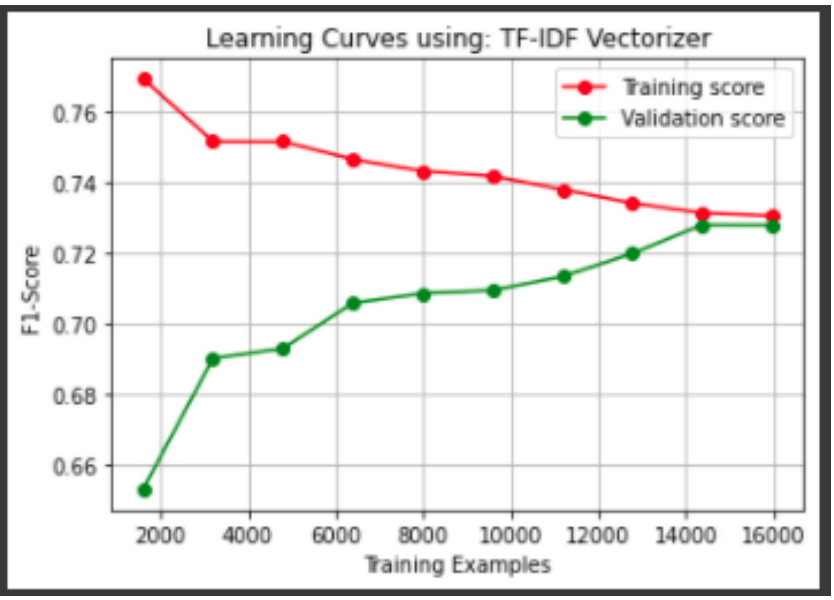
- Εφαρμόζοντας την εξής προεπεξεργασία στα δεδομένα:
  - Μετατροπή του κειμένου σε lower case
  - Αφαίρεση links, mentions και αριθμών
  - Αντικατάσταση των emojis με το αντίστοιχο text
  - Αφαίρεση συμβόλων και σημείων στίξης
- Χρησιμοποιώντας τον TF-IDF Vectorizer, με τις εξής παραμέτρους:
  - min_df = 1 (default)
  - max_df = 1.0 (default)
  - max_features = 100.000
  - n_gram = (1,3)
- Χρησιμοποιώντας τον SVD για την μείωση των features με την παράμετρο:
  - n_components=600
- Τέλος, χρησιμοποιώντας τον softmax με τις εξής παραμέτρους:
  - solver = newton-cg
  - C = 3
  - max_iter = 1000

Τα αποτελέσματα της μεθόδου είναι τα εξής:

```
Softmax Regression using vectorizer: TF-IDF
 Accuracy: 73.14%
 f1 score: 72.79%
 Precision: 73.75%
 Recall: 73.14%
```



	precision	recall	f1-score	support
0	0.82	0.75	0.78	1065
1	0.68	0.44	0.53	296
2	0.67	0.80	0.73	921
accuracy			0.73	2282
macro avg	0.72	0.66	0.68	2282
weighted avg	0.74	0.73	0.73	2282



Συμπεράσματα

Προεπεξεργασία

Δοκιμάζοντας όλες τις τεχνικές προεπεξεργασίας των δεδομένων που αναφέρθηκαν παρατήρησα τα εξής:

- Περίπλοκες τεχνικές φαίνεται να είναι μη αποδοτικές με αποτέλεσμα να μειώνεται το συνολικό score που επιτυγχάνει το μοντέλο μας. Εφαρμόζοντας τις Stemming και Lemmatization το μοντέλο δεν παρουσιάζει καλή απόδοση. Επίσης, πολύ διαδεδομένες τεχνικές όπως η αφαίρεση των stopwords, φάνηκε μη αποδοτική για το δεδομένο dataset.
- Σε κάθε περίπτωση η συνολική απόδοση του μοντέλου αυξήθηκε ελάχιστα σε σχέση με τις δοκιμές που έκανα χωρίς να εφαρμόσω καθόλου προεπεξεργασία στα δεδομένα.
- Άρα καταλήγουμε στο ότι η προεπεξεργασία των δεδομένων για το δοθέν dataset δεν αποδείχθηκε πολύ αποδοτική.

Vectorization

Έπειτα από όλες τις διαφορετικές δοκιμές που αναφέρθηκαν, αλλά και ακόμα περισσότερες που έκανα αλλά δεν αναφέρθηκαν εδώ, κατέληξα πως ο αποδοτικότερος vectorizer είναι ο TF-IDF.
Όπως φαίνεται και από τα αποτελέσματα των δοκιμών που παρουσίασα ο TF-IDF έχει την καλύτερη απόδοση σε κάθε μια από τις δοκιμές ανεξαιρέτως.
Όσον αφορά τις παραμέτρους που επέλεξα:

- max_features = 100.00 , επέλεξα ο TF-IDF να παράγει αρκετά features τα οποία να μειώνονται στην συνέχεια μέσω του SVD. Μειώνοντας την τιμή του max_features αγνοούνταν σημαντικά features με αποτέλεσμα να πέφτει η απόδοση, ενώ αυξάνοντας την κρατούσε «άχρηστα» features τα οποία δεν έδιναν χρήσιμες πληροφορίες με αποτέλεσμα και πάλι να πέφτει η απόδοση. Έτσι κατέληξα στην τιμή 100.000 με την οποία παρατήρησα την βέλτιστη απόδοση.
- n_gram = (1,3) , δοκιμάζοντας όλους τους συνδυασμούς: (1,1),(1,2),(2,2),(1,3),(2,3) παρατήρησα την καλύτερη απόδοση χρησιμοποιώντας την παράμετρο (1,3) δηλαδή κρατώντας unigrams, bigrams και trigrams. Ωστόσο, η καλή απόδοση που παρατήρησα με το (1,3) συνδέεται άμεσα με την κατάλληλη επιλογή του max_features. Με την παράμετρο (1,3) παράγονται συνολικά πάρα πολλά features με αποτέλεσμα πολλά από αυτά να είναι «άχρηστα». Μια λάθος επιλογή του max_feature σε συνδυασμό με το n_gram=(1,3) μπορεί να οδηγήσει σε κατακόρυφη πτώση της απόδοσης. Συνδυάζοντας τις δύο αυτές τιμές κατάλληλα, έπειτα από πολλές δοκιμές, κατάφερα να πετύχω την απόδοση που παρουσιάστηκε.

Singular Value Decomposition (SVD)

Χρησιμοποιώντας τον SVD για την ελαχιστοποίηση των features κάθε αναπαράστασης παρατήρησα βελτίωση της απόδοσης, για αυτό και επιλέχθηκε.
Πιο συγκεκριμένα:

- Χρησιμοποιώντας τον SVD παρατηρούμε αύξηση του score που επιτυγχάνει το μοντέλο μας ενώ ταυτόχρονα εξαλείφεται τελείως το overfit που παρουσίαζε το μοντέλο μας.
- Ωστόσο, προσθέτει επιπλέον χρόνο στην συνολική εκτέλεση καθώς η διαδικασία μετατροπής των features είναι αρκετά κοστοβόρα.
- Τέλος, παρατηρούμε διαφορετικά scores σε κάθε εκτέλεση λόγω της τυχαιότητας του SVD. Ωστόσο σε κάθε περίπτωση τα αποτελέσματα αυτά είναι αρκετά καλύτερα σε σχέση με αυτά που παρατηρήθηκαν με την Μείωση των features μέσω των παραμέτρων των vectorizers.

Softmax

Όπως έχει ήδη αναφερθεί, για την υλοποίηση του softmax χρησιμοποίησα το μοντέλο Logistic Regression ορίζοντας την παράμετρο multi_class='multinomial'.
Όσον αφορά τις υπόλοιπες παραμέτρους που επέλεξα:

- solver = newton-cg , δοκιμάζοντας όλους τους δυνατούς solvers παρατήρησα τα εξής:
    - lbfgs : αρκετά αργός καθώς είναι ιδανικός για αρκετά μικρά/απλά datasets. Επίσης τα αποτελέσματα που έβγαζε δεν ήταν ικανοποιητικά.
    - Sag και saga : πολύ γρήγοροι καθώς χρησιμοποιούνται για μεγάλα datasets. Ωστόσο η ακρίβεια που επιτυγχανόταν δεν ήταν τόσο καλή καθώς υστερούσε σε σχέση με τον newton-cg.
    - newton-cg : όχι ο γρηγορότερος ωστόσο η καθυστέρηση για το δοθέν dataset ήταν αμελητέα. Επίσης μέσω αυτού επιτυγχάνονταν τα καλύτερα αποτελέσματα από όλους τους υπόλοιπους solvers, για αυτό και επιλέχθηκε.
- C = 3 , δοκίμασα όλες τις τιμές εντός του range [1,10] καθώς και κάποιες μεγαλύτερες. Τα καλύτερα αποτελέσματα τα πέτυχα επιλέγοντας την τιμή 3.
- max_iter = 1000 , επέλεξα να αυξήσω την τιμή των μέγιστων επαναλήψεων έτσι ώστε σε κάθε περίπτωση να συγκλίνει ο αλγόριθμος και να πετυχαίνουμε την βέλτιστη απόδοση του μοντέλου.

Γενικά

Λαμβάνοντας υπόψιν το δοθέν dataset εκπαίδευσης και εφαρμόζοντας όλες τις τεχνικές που περιέχονται στο πλαίσιο της εργασίας, το τελικό μοντέλο που επιτεύχθηκε είναι αρκετά ικανοποιητικό και (με βάση όσα παρατήρησα) τα περιθώρια για περαιτέρω βελτίωση είναι πολύ μικρά.
Εφαρμόζοντας τεχνικές όπως word2Vec, ή χρησιμοποιώντας πιο σύνθετα μοντέλα ενδεχομένως να μπορούν αυξήσουν την ακρίβεια του μοντέλου, ωστόσο ξεφεύγουν από τα πλαίσια αυτής της εργασίας.