Τεχνητή Νοημοσύνη 2
(Homework 1)

Ονοματεπώνυμο: Δημήτριος Σιταράς
Αριθμός μητρώου: 1115201800178
Εξάμηνο: 7ο

Περιεχόμενα

Άσκηση 1 2
Vaccine Sentiment Classifier Using SoftMax Regression 5
Γενικά 5
Μοντέλα 5
Παρατηρήσεις 8

Άσκηση 1

Αρκεί να δείξω ότι:

$$\nabla_{\mathrm{w}} MSE(\mathrm{w}) = \frac{2}{\mathrm{m}} \left(\mathrm{X}^{\mathrm{T}}(X \cdot w - \mathrm{y})\right)$$

Αρχικά, έχω:

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

$$= \frac{1}{\mathrm{m}} \sum_{i=1}^{m} (\, w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_n x_n^{(i)} - y^{(i)})^2$$

Έστω, ότι $w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots + w_n x_n^{(i)} - y^{(i)} = u_i(w)$, τότε αν πάρω την μερική παράγωγο ως προς w_1 θα έχω:

$$\frac{\vartheta}{\vartheta w_1} MSE(w) = \frac{\vartheta}{\vartheta w_1} \frac{1}{\mathrm{m}} \sum_{i=1}^{m} (u_i(w))^2 = \frac{1}{\mathrm{m}} \sum_{i=1}^{m} \frac{\vartheta}{\vartheta w_1} (u_i(w))^2 =$$

$$= \frac{1}{\mathrm{m}} \sum_{i=1}^{m} 2u_i(w) \frac{\vartheta}{\vartheta w_1} u_i(w)$$

Όπου,

$$\frac{\vartheta}{\vartheta w_1} u_i(w) = \frac{\vartheta}{\vartheta w_1} w_1 x_1^{(i)} + \frac{\vartheta}{\vartheta w_1} w_2 x_2^{(i)} + \ldots + \frac{\vartheta}{\vartheta w_1} w_n x_n^{(i)} - \frac{\vartheta}{\vartheta w_1} y^{(i)} = x_1^{(i)}$$

Άρα, $\frac{\vartheta}{\vartheta w_1} MSE(w) = \frac{2}{\mathrm{m}} \sum_{i=1}^{m} u_i x_1^{(i)}$

Συνεπώς, για το $\nabla_{\mathrm{w}} MSE(\mathrm{w})$ θα έχω:

$$\nabla_{\mathrm{w}} MSE(\mathrm{w}) = \begin{pmatrix} \frac{\vartheta}{\vartheta w_1} MSE(w) \\ \vdots \\ \frac{\vartheta}{\vartheta w_n} MSE(w) \end{pmatrix} = \begin{pmatrix} \frac{2}{\mathrm{m}} \sum_{i=1}^{m} u_i x_1^{(i)} \\ \vdots \\ \frac{2}{\mathrm{m}} \sum_{i=1}^{m} u_i x_n^{(i)} \end{pmatrix} =$$

$$= \frac{2}{\mathrm{m}} \begin{pmatrix} \sum_{i=1}^{m} \left(w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_n x_n^{(i)} - y^{(i)}\right) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} \left(w_1 x_1^{(i)} + w_2 x_2^{(i)} + \cdots + w_n x_n^{(i)} - y^{(i)}\right) x_n^{(i)} \end{pmatrix} =$$

$$= \frac{2}{\mathrm{m}} \begin{pmatrix} \sum_{i=1}^{m} \left(w x^{(i)} - y^{(i)}\right) x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} \left(w x^{(i)} - y^{(i)}\right) x_n^{(i)} \end{pmatrix}$$

Στη συνέχεια, ορίζω τους πίνακες X, X^T, w, y και κάνω πράξεις υπολογίζοντας το $\mathrm{X}^{\mathrm{T}}(X \cdot w - \mathrm{y})$, ώστε να καταλήξω στο παραπάνω, οπότε έχω:

$$\mathrm{X} = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix}, \quad w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, \quad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

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

$$X \cdot w = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} x^{(1)} \cdot w \\ \vdots \\ x^{(m)} \cdot w \end{bmatrix}$$

Επομένως,

$$\mathrm{X}^{\mathrm{T}}(X \cdot w - \mathrm{y}) = \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \left(\begin{bmatrix} x^{(1)} \cdot w \\ \vdots \\ x^{(m)} \cdot w \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \right) =$$

$$= \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \ddots & \vdots \\ x_n^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} x^{(1)} \cdot w - y^{(1)} \\ \vdots \\ x^{(m)} \cdot w - y^{(m)} \end{bmatrix} =$$

$$= \begin{bmatrix} x_1^{(1)}\left(x^{(1)} \cdot w - y^{(1)}\right) + x_1^{(2)}\left(x^{(2)} \cdot w - y^{(2)}\right) + \cdots + x_1^{(m)}\left(x^{(m)} \cdot w - y^{(m)}\right) \\ \vdots \\ x_n^{(1)}\left(x^{(1)} \cdot w - y^{(1)}\right) + x_n^{(2)}\left(x^{(2)} \cdot w - y^{(2)}\right) + \cdots + x_n^{(m)}\left(x^{(m)} \cdot w - y^{(m)}\right) \end{bmatrix}$$

$$= \begin{pmatrix} \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_n^{(i)} \end{pmatrix}$$

Πολλαπλασιάζοντας και με τον όρο $\frac{2}{\mathrm{m}}$, καταλήγω στο εξής:

$$\frac{2}{\mathrm{m}}\left(\mathrm{X}^{\mathrm{T}}(X \cdot w - \mathrm{y})\right) = \frac{2}{\mathrm{m}} \begin{pmatrix} \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_n^{(i)} \end{pmatrix}$$

Όπως έδειξα παραπάνω έχω ότι:

$$\nabla_{\mathrm{w}} MSE(\mathrm{w}) = \frac{2}{\mathrm{m}} \begin{pmatrix} \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} (wx^{(i)} - y^{(i)})\, x_n^{(i)} \end{pmatrix}$$

Άρα, τελικά:

$$\nabla_{\mathrm{w}} MSE(\mathrm{w}) = \frac{2}{\mathrm{m}}\left(\mathrm{X}^{\mathrm{T}}(X \cdot w - \mathrm{y})\right)$$

Vaccine Sentiment Classifier Using SoftMax Regression

Γενικά

Αρχικά, στην υλοποίηση μου κατασκευάζω δύο «διαφορετικά» train sets (επομένως και δύο «διαφορετικά» validation sets), στο πρώτο train και validation set έχω τα tweets, αυτούσια από τα αρχεία .csv χωρίς την εφαρμογή preprocessing στα αντίστοιχα data frames. Ενώ, στο δεύτερο train και validation set έχω «καθαρίσει/προ-επεξεργαστεί» τα tweets, αφαιρώντας από αυτά τα URLs, τα mentions, τα emojis, τα σημεία στίξης και οποιαδήποτε σύμβολα (όλα αυτά γίνονται μέσω της συνάρτησης cleaning(…)). Μάλιστα, έχω εφαρμόσει stemming και lemmatization, μέσω των αντίστοιχων συναρτήσεων. Επιπλέον, δοκίμασα διάφορους συνδυασμούς και πειραματισμούς στο preprocessing, για παράδειγμα: δοκίμασα να χρησιμοποιήσω μόνο την συνάρτηση cleaning (χωρις stemming και lemmatization), δοκίμασα να κάνω μόνο stemming στα tweets, δοκίμασα να αφαιρέσω «τα πάντα» εκτός των emojis (με την σκέψη ότι ίσως σε κάποια tweets εκφράζουν κάτι) κλπ. Όμως, βλέποντας την τελική αξιολόγηση (scores αλλά και learning curve) του μοντέλου που πρόκυπτε κάθε φορά, κατέληξα πως το βέλτιστο είναι να τα εφαρμόσω όλα μαζί (cleaning, stemming, lemmatization) πετυχαίνοντας καλύτερα scores και learning curves στα αντίστοιχα μοντέλα σε σχέση με όλες τι υπόλοιπες δοκιμές μου.
Αποθηκεύω, λοιπόν, τα tweets του train και του validation set που ΔΕΝ έχουν δεχθεί προ-επεξεργασία και αυτά που έχουν σε 4 διαφορετικές λίστες (tweetsTrain και tweetsVal, tweetsTrainPreprocessing και tweetsValPreprocessing). Τα αντίστοιχα σύνολα που έχουν label του train και του validation είναι κοινά και στις 2 περιπτώσεις οπότε αποθηκεύονται σε 2 λίστες (labelsTrain και labelsVal).

Μοντέλα

Στη συνέχεια, με βάση των λιστών tweetsTrain και tweetsTrainPreprocessing κατασκευάζω τους αντίστοιχους πίνακες με αναπαράσταση BagOfWords, Td-idf και Hashing για τα tweets των δυο «διαφορετικών» train sets, δημιουργώ 2 ίδιους SoftMax Classifiers, ώστε να τους εφαρμόσω ξεχωριστά στα δυο αυτά train sets. Έπειτα, λοιπόν, πραγματοποιώ classification με τις ακόλουθες αναπαραστάσεις: BoW, Td-idf και Hashing, έχοντας ως Classifier τον SoftMax. Έτσι, σε κάθε αναπαράσταση ουσιαστικά έχω δύο διαφορετικά μοντέλα: το πρώτο κάθε φορά εκπαιδεύεται με το train set tweetsTrain, ενώ το δεύτερο με το train set tweetsTrainPreprocessing. Τελικά, κάθε μοντέλο κάνει prediction των labels του αντίστοιχου validation set. Οι Vectorizers που χρησιμοποιώ με τις βέλτιστες παραμέτρους για την κατασκευή των πινάκων είναι οι εξής:

- **CountVectorizer** με παραμέτρους:

    - *max_df=0.95*: ώστε να αγνοήσει τις λέξεις που εμφανίζονται περισσότερο από το 95% στο σύνολο των tweets. Δοκίμασα και πιο μεγάλα ποσοστά χωρις να δω κάποια βελτιστοποίηση. Ωστόσο, για τα πιο μικρά ποσοστά παρατήρησα πως αγνοούνται παρα πάνω λέξεις με αποτέλεσμα να αλλοιώνονται τα αποτελέσματα του μοντέλου προς το χειρότερο.

    - *min_df=0.0025*: ώστε να αγνοήσει τις λέξεις που εμφανίζονται λιγότερο από το 0.25% στο σύνολο των tweets. Γενικά, παρατήρησα πως στο εύρος τιμών από 0.0020 μέχρι 0.0050 έπαιρνα τα καλύτερα scores και learning curves. Τελικά, με πολλούς πειραματισμούς κατέληξα στο παραπάνω. Έτσι, καταφέρνω να μειώσω τον αριθμό των features, διασφαλίζοντας πως το μοντέλο δεν θα κάνει overfit.

    - *ngram_range=(1,2):* ώστε να «σπάει» τις προτάσεις σε καθε tweet σε unigrams και bigrams. Δοκίμασα, επίσης, (1,1) ώστε να έχω μόνο unigrams και (2,2) ώστε να έχω μόνο bigrams, αλλα τα τελικά αποτελέσματα ήταν αρκετά χειρότερα.

    - Επίσης, δοκίμασα να παραλείψω τα stop words βάζοντας στον vectorizer την παράμετρο *stop_words='english',* όμως τα scores του μοντέλου που πρόκυπτε ήταν χειρότερα.

    - Τέλος, παρατήρησα πως για το train set (χωρίς preprocessing): tweetsTrain, αν δεν χρησιμοποιήσω τις παραμέτρους max_df και min_df (ώστε να μειώσω τον αριθμό των features), έχοντας μόνο *ngram_range=(1,2),* πετυχαίνω μεγαλύτερη ακρίβεια στις προβλέψεις και γενικά καλύτερα scores με το μοντέλο που προκύπτει, αλλά η αντίστοιχη learning curve δείχνει ότι μοντέλο κανει overfitting, το οποίο θέλουμε να αποφύγουμε, καθώς σε αυτή την περίπτωση το μοντέλο έχει μάθει το train set τόσο καλά που δεν μπορεί να γενικευτεί σε νέα δεδομένα.

- **TfidfVectorizer** με παραμέτρους:

    - *max_df=0.95*: ώστε να αγνοήσει τις λέξεις που εμφανίζονται περισσότερο από το 95% στο σύνολο των tweets. Δοκίμασα και πιο μεγάλα ποσοστά χωρις να δω κάποια βελτιστοποίηση. Ωστόσο, για τα πιο μικρά ποσοστά παρατήρησα πως αγνοούνται παρα πάνω λέξεις με αποτέλεσμα να αλλοιώνονται τα αποτελέσματα του μοντέλου προς το χειρότερο.

    - *min_df=0.0026*: ώστε να αγνοήσει τις λέξεις που εμφανίζονται λιγότερο από το 0.26% στο σύνολο των tweets. Γενικά, παρατήρησα πως στο εύρος τιμών από 0.0020 μέχρι 0.0050 έπαιρνα τα καλύτερα scores και

learning curves. Τελικά, με πολλούς πειραματισμούς κατέληξα στο παραπάνω. Έτσι, καταφέρνω να μειώσω τον αριθμό των features, διασφαλίζοντας πως το μοντέλο δεν θα κάνει overfit.

- *ngram_range=(1,2):* ώστε να «σπάει» τις προτάσεις σε καθε tweet σε unigrams και bigrams. Δοκίμασα, επίσης, (1,1) ώστε να έχω μόνο unigrams και (2,2) ώστε να έχω μόνο bigrams, αλλα τα τελικά αποτελέσματα ήταν αρκετά χειρότερα.

- Επίσης, δοκίμασα να παραλείψω τα stop words βάζοντας στον vectorizer την παράμετρο *stop_words='english',* όμως τα scores του μοντέλου που πρόκυπτε ήταν χειρότερα.

- Τέλος, παρατήρησα πως για το train set (χωρίς preprocessing):: tweetsTrain, αν δεν χρησιμοποιήσω τις παραμέτρους max_df και min_df (ώστε να μειώσω τον αριθμό των features), έχοντας μόνο *ngram_range=(1,2),* πετυχαίνω μεγαλύτερη ακρίβεια στις προβλέψεις και γενικά καλύτερα scores με το μοντέλο που προκύπτει, αλλά η αντίστοιχη learning curve δείχνει ότι μοντέλο κανει overfitting, το οποίο θέλουμε να αποφύγουμε, καθώς σε αυτή την περίπτωση το μοντέλο έχει μάθει το train set τόσο καλά που δεν μπορεί να γενικευτεί σε νέα δεδομένα.

- **HashingVectorizer** με παραμέτρους:

  - n_features=2**10: πειραματίστηκα με διάφορα μεγέθη χαρακτηριστικών, αν βάλω πολύ μικρό μέγεθος (π.χ. 2**2) τότε το αντίστοιχο μοντέλο παρουσιάζει underfitting, ενώ αν βάλω πολύ μεγάλο μέγεθος το μοντέλο παρουσιάζει overfitting. Συνεπώς κατέληξα σε κάτι ενδιάμεσο, αποφεύγοντας αυτές τις συμπεριφορές.

  - *ngram_range=(1,2):* ώστε να «σπάει» τις προτάσεις σε καθε tweet σε unigrams και bigrams. Δοκίμασα, επίσης, (1,1) ώστε να έχω μόνο unigrams και (2,2) ώστε να έχω μόνο bigrams, αλλα τα τελικά scores του μοντέλου ήταν αρκετά χειρότερα.

  - Επίσης, δοκίμασα να παραλείψω τα stop words βάζοντας στον vectorizer την παράμετρο *stop_words='english',* όμως τα scores του μοντέλου που πρόκυπτε ήταν χειρότερα.

Συνεπώς, κατασκευάζω 6 μοντέλα εφαρμόζοντας σε όλα ουσιαστικά τον classifier SoftMax, που ζητείται από την εκφώνηση, τον οποίο για να τον κατασκευάσω χρησιμοποίησα την LogisticRegression(...) με τις εξής παραμέτρους:

- multi_class="multinomial": η συγκεκριμένη παράμετρος υλοποιεί ουσιαστικά τον SoftMax classifier.

- solver="newton-cg": επέλεξα να χρησιμοποιήσω αυτόν τον solver γιατί είναι ακριβής στους υπολογισμούς του σε σχέση με τους υπόλοιπους (lbfgs, sag, saga) που υποστηρίζουν την multinomial logistic regression ή απλά SoftMax. Επίσης, είναι σχετικά γρήγορος. Οι solvers sag και saga αν και ειναι οι πιο γρήγοροι δεν βγάζουν τόσο ακριβή αποτελέσματα, συνήθως χρησιμοποιούνται σε πολύ μεγάλα dataset. Ο solver lbfgs είναι σχετικά αργός και επίσης δεν βγάζει τόσο ακριβή αποτελέσματα για μεσαία προς μεγάλα dataset, όπως ειναι αυτό που έχουμε, είναι ιδανικός για μικρά.

- max_iter=1000: ο μέγιστος αριθμός επαναλήψεων που θα κανει ο solver για να συγκλίνει. Παρατήρησα πως για το dataset μας ο solver newton-cg σίγουρα θα καταφέρει να συγκλίνει μέχρι τις 1000 επαναλήψεις.

- C=10.0: δοκίμασα διάφορες τιμές όπως 1.0 (default), 5.0, 15.0, 20.0 καταλήγοντας πως η τιμή 10.0 είναι η καλύτερη.

Παρατηρήσεις

Στο τέλος της υλοποίησης μου, παρουσιάζω για κάθε ένα από τα 6 μοντέλα, τα αντίστοιχα αποτελέσματα του όσον αναφορά το accuracy,το precision, το recall, το f1-score, το support καθώς και το learning curve.
Συγκεκριμένα, έχω τα εξής μοντέλα:

1) Classification με αναπαράσταση **Bag of Words** και εκπαίδευση του μοντέλου στο train set: **tweetsTrain**.

2) Classification με αναπαράσταση **Bag of Words** και εκπαίδευση με το μοντέλου στο train set: **tweetsTrainPreprocessing**.

3) Classification με αναπαράσταση **Tf-idf** και εκπαίδευση του μοντέλου με το train set: **tweetsTrain**.

4) Classification με αναπαράσταση **Tf-idf** και εκπαίδευση του μοντέλου με το train set: **tweetsTrainPreprocessing**.

5) Classification με αναπαράσταση **Hashing** και εκπαίδευση του μοντέλου με το train set: **tweetsTrain**.

6) Classification με αναπαράσταση **Hashing** και εκπαίδευση του μοντέλου με το train set: **tweetsTrainPreprocessing**.

Γενικά, συμπεραίνω ότι (και από τις δοκιμές που έκανα αλλά και τα τελικά scores των έξι αυτών μοντέλων που παρουσιάζω) πως ένα μοντέλο ανεξάρτητα της αναπαράστασης που χρησιμοποιείται «βγάζει» συνήθως, για το συγκεκριμένο dataset, καλύτερα αποτελέσματα όταν ο SoftMax classifier εκπαιδεύεται πάνω στο train set: **tweetsTrain,** στο οποίο δεν πραγματοποιείται preprocessing στα tweets. Οπότε, τις περισσότερες φορές το preprocessing δεν οδηγεί σε καλύτερα αποτελέσματα. Βέβαια, η διαφορά των score που παρατηρείται μεταξύ των μοντέλων που εκπαιδεύτηκαν πάνω στο train set **tweetsTrain** και των μοντέλων που εκπαιδεύτηκαν πάνω στο train set **tweetsTrainPreprocessing** είναι ελάχιστη. Βλέποντας, επίσης, τα scores αλλά και τις αντίστοιχες learning curves του κάθε μοντέλου μπορώ να κατατάξω τους vectorizers από τον καλύτερο προς τον χειρότερο: TfidfVectorizer, CountVectorizer και HashingVectorizer.

Τέλος, καταλήγω πως το καλύτερο από τα έξι μοντέλα που κατασκεύασα με βάση τα scores (accuracy: 71.12%, precision: 71%, recall: 71%, f1-score: 71%) και τα learning curves είναι **το 3ο**, δηλαδή αυτό το μοντέλο που χρησιμοποιεί για την αναπαράσταση τον TF-IDF vectorizer και έχει εκπαιδευτεί στο train set (που δεν έχει «υποστεί» preprocessing): **tweetsTrain.**
Επομένως, «διαλέγω» αυτό το μοντέλο για να προβλέψει τα labels του test set που θα δώσει. Αρκεί, λοιπόν να αντικατασταθεί το path του validation set με το path του test set στο παρακάτω σημείο:

```
1 Location = r'vaccine_train_set.csv'
2 trainDf=pd.read_csv(Location)
3 Location = r'vaccine_validation_set.csv' # replace this one with the path of the test set
4 validationDf=pd.read_csv(Location)
```

Και στην συνέχεια να εκτελεστούν τα αντίστοιχα cells στο Jupiter Notebook που υλοποιούν το Classification με αναπαράσταση Tf-idf και εκπαίδευση του μοντέλου με το train set: **tweetsTrain** (δηλαδή χωρίς preprocessing).