Artificial Intelligence II

Nefeli Tavoulari

Fall Semester 2021

1

The Mean Squared Error loss function is defined as:

$\mathcal{MSE} = \frac{1}{m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$

Calculating the gradient of MSE, we can find out the parameters which minimize the loss.

$\nabla_{\mathbf{w}} \mathcal{MSE} = \nabla_{\mathbf{w}} \frac{1}{m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$

The features x_i and the predicted values y_i can be considered as constants, since the gradient is with respect to w, which is the model's parameter vector.
(scalar multiplication rule)

$\nabla_{\mathbf{w}} \mathcal{MSE} = \frac{1}{m} \nabla_{\mathbf{w}} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$

Then, assuming we only have one instance (x,y) and using the power rule and the chain rule for derivatives:

$\nabla_{\mathbf{w}} \mathcal{MSE} = \nabla_{\mathbf{w}} (h_w(x) - y)^2$

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(h_w(x) - y)\nabla_{\mathbf{w}}(h_w(x) - y)$

where $h_w(x) = w_0x_0 + ... + w_nx_n = w^Tx$: **(*)**

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(h_w(x) - y)\nabla(w_0x_0 + ... + w_nx_n - y)$

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(h_w(x)-y)(\frac{\partial(w_0x_0+...+w_nx_n-y)}{\partial w_0}, ..., \frac{\partial(w_0x_0+...+w_nx_n-y)}{\partial w_n})$

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(h_w(x) - y)(x_0, ..., x_n)$

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(h_w(x) - y)(x)$

so, from **(*)**:

$\nabla_{\mathbf{w}}\mathcal{MSE} = 2(w^Tx - y)(x)$

which equals to the following vector:

$$\begin{bmatrix} 2(w^Tx - y)(x_0) \\ ... \\ 2(w^Tx - y)(x_n) \end{bmatrix}$$

Now, for m training instances:

$\nabla_{\mathbf{w}}\mathcal{MSE} = \frac{2}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)\nabla_{\mathbf{w}}(h_w(x_i) - y_i)$

$$\nabla_{\mathbf{w}}\mathcal{MSE} = \begin{bmatrix} \frac{2}{m}(w^Tx_1 - y_1)(x_{1,0}) + ... + \frac{2}{m}(w^Tx_m - y_m)(x_{m,0}) \\ ... \\ \frac{2}{m}(w^Tx_1 - y_1)(x_{1,n}) + ... + \frac{2}{m}(w^Tx_m - y_m)(x_{m,n}) \end{bmatrix}$$

So: $\nabla_{\mathbf{w}}\mathcal{MSE} = \frac{2}{m}(\mathrm{X}^T(\mathrm{Xw - y}))$.

2 Vaccine Sentiment Classification

In this notebook I trained a multinomial(softmax) Logistic Regression model, which classifies tweets as Neutral, Pro-vax or Anti-vax.

2.1 Dataset

The given dataset consists of two columns. The one column represents the tweets text and the other one the corresponding label (0, 1, 2 / neutral,anti-vax,pro-vax). Also, the dataset is rather imabalanced, since the anti-vax instances are almost 3 times fewer ,in comparison with the other classes. One conclusion we might reach out of it, is the fact that accuracy is probably not the right metric to evaluate our model.

2.2 Preprocessing

First of all, I performed some preprocessing and feature extraction on the data. I removed all empty or duplicate tweets, as they offer no useful information to the model, I lowercased tweets, performed lemmatization, removed stopwords, special characters, emojis, words written in a non-latin alphabet, urls and twitter accounts, in order for the model to focus on important and linguistically meaningful words, instead of noise. Also, I experimented with stemmers :

- Porter (removes common endings of words)
- Lancaster (more aggressive)
- Snowball (better version of Porter)

but I noticed that the Wordnet Lemmatizer performed much better than all of them, so I have commented out the lines that correspond to the stemmers in the notebook. In general, what I understood by my research is that lemmatization provides better results at times, because it performs a vocabulary and morphological analysis of the words and produces real, dictionary words, in contrast with stemming. The results I took from Stemming and Lemmatization, using CountVectorizer with TfidfTransformer and multinomial Logistic Regression on training and validation data:

	Porter	*Lancaster*	*Snowball*	*Lemmatizer*
Recall	0.8214	0.8165	0.8210	**0.8377**
Precision	0.8214	0.8165	0.8210	**0.8377**
$F1$	0.8214	0.8165	0.8210	**0.8377**
Accuracy	0.8214	0.8165	0.8210	**0.8377**

	Porter	*Lancaster*	*Snowball*	*Lemmatizer*
Recall	0.7094	0.7085	0.7112	**0.7177**
Precision	0.7094	0.7085	0.7112	**0.7177**
$F1$	0.7094	0.7085	0.7112	**0.7177**
Accuracy	0.7094	0.7085	0.7112	**0.7177**

We observe here that the Snowball Stemmer really is a better version of the Porter Stemmer. The Lemmatizer is the best solution though and so, from now on, this is the one I will be using.

2.3 Feature Extraction

Then, I vectorized and calculated TF-IDF on the textual data, so as to perform Logistic Regression. For this task, I tried out :

- CountVectorizer with TfidfTransformer, (computes word counts, IDF values and TF-IDF scores)
- HashingVectorizer with TfidfTransformer, (uses hashing to find the token string name to feature integer index mapping)
- TfidfVectorizer (CountVectorizer + TfidfTransformer)

An advantage of HashingVectorizer is that it does not store the resulting vocabulary and so, it is very low memory cost. However, there can be collisions, because of string hashing, which is not a problem if the number of features is large enough.The results on both training and data using multinomial Logistic Regression:

	$CountVec$	$HashingVec$	$TfidfVec$
$Recall$	0.8377	0.8287	0.8377
$Precision$	0.8377	0.8287	0.8377
$F1$	0.8377	0.8287	0.8377
$Accuracy$	0.8377	0.8287	0.8377

	$CountVec$	$HashingVec$	$TfidfVec$
$Recall$	0.7177	0.7169	0.7177
$Precision$	0.7177	0.7169	0.7177
$F1$	0.7177	0.7169	0.7177
$Accuracy$	0.7177	0.7169	0.7177

We observe that the HashingVectorizer did not perform as well as the others. We also observe that the model might be overfitting, since the training score is quite higher than the validation score. Thus, I performed some hyperparameter tuning, trying plenty of different combinations in order to improve the performance and avoid overfitting.

For the **CountVectorizer**, I added max_df and min_df, in order to exclude some extremely rare or extremely used words, since they offer nothing but noise to the model. For the same reason, I also added the stopwords option (even though I had already done it manually in preprocessing). In this way, words, such as "the" or "asdfgh" are removed and that improves the performance of the model. Also, I added some regularization to the Logistic Regression model(C=1.5), in order to tell the model that the data might not be fully representative of the real world data.

For the **HashingVectorizer**, I added the stopwords option and enabled unigrams and bigrams also. Moreover, I added some regularization to the Logistic Regression model(C=1.9), elasticnet penalty, saga solver and l1_ratio. Elastic net regression is a combination of ridge and lasso regression. It shrinks some coefficients towards zero (like ridge regression) and set some coefficients to exactly zero (like lasso regression). Saga solver performs well for sparse multinomial logistic regression,it allows for L1 regularization and it is ,in general, faster

than sag. L1 ratio determines whether the penalty is a combination of L1 and L2. Finally, I set the number of features equal to 2^{16}, because that gave me the higher score, probably because it avoids collision).

For the **TfIdfVectorizer**, I added the same parameters as for the CountVectorizer.

In this way, I handled overfitting.After hyperparameter tuning:

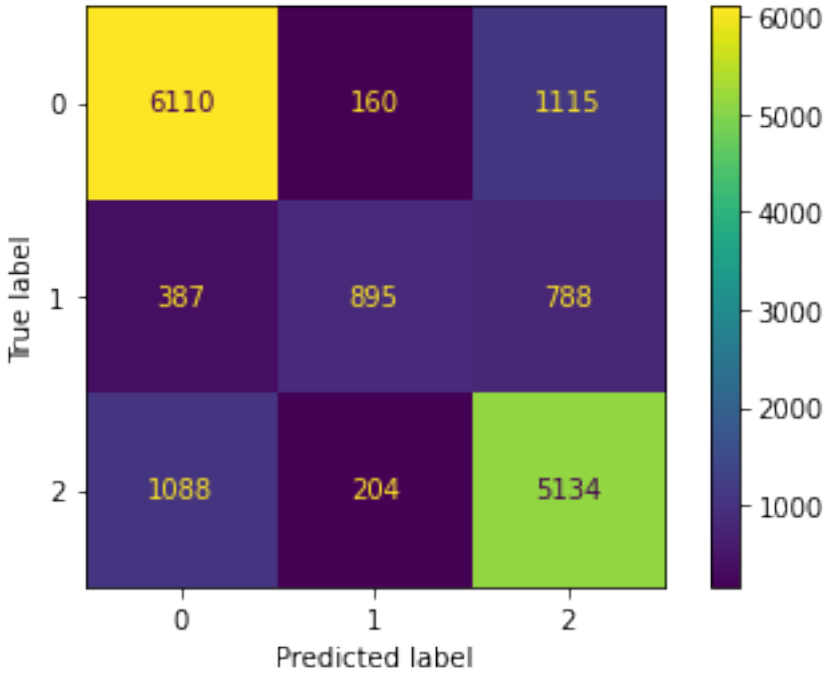
	$CountVec$	$HashingVec$	$TfidfVec$
$Recall$	0.7643	**0.8030**	0.7643
$Precision$	0.7643	**0.8030**	0.7643
$F1$	0.7643	**0.8030**	0.7643
$Accuracy$	0.7643	**0.8030**	0.7643

	$CountVec$	$HashingVec$	$TfidfVec$
$Recall$	0.7230	**0.7353**	0.7230
$Precision$	0.7230	**0.7353**	0.7230
$F1$	0.7230	**0.7353**	0.7230
$Accuracy$	0.7230	**0.7353**	0.7230

In general we observe that the training scores got worse than before hyperparameter tuning, but the validation scores, which we are more interested in, were improved. After all, the Hashing Vectorizer performs better than the others, but there is a good chance that this model is still overfitting.

2.4 Evaluation

Using different metrics, precision, recall, f1 score, accuracy, confusion matrix, roc auc, I was able to determine the quality of my model. From the confusion matrix, I understand that my model makes good enough predictions, since the diagonal has quite bigger numbers. I observe also that it does not predict that well the anti-vaxers, but that is expected, because the anti-vaxers in the data are very few, in comparison to the other classes. The confusion matrix ,using the Count Vectorizer :



Also, I would like to note that in a multi-class classification setup, micro-average is preferable if you suspect there might be class imbalance. Furthermore, micro-F1 = micro-precision = micro-recall = accuracy and TfIdfVectorizer has the same performance as the CountVec-

torizer with TfIdfTransformer. So, our results seem normal. As far as the ROC AUC is concerned, it is a metric that tells us how much the model is capable of distinguishing between classes.
The results I took were the following:

	$CountVec$	$HashingVec$	$TfidfVec$
$OvO - AUC - macro$	0.846171	**0.851729**	0.846171
$OvO - AUC - weighted$	0.850188	**0.856657**	0.850188
$OvR - AUC - macro$	0.855235	**0.862937**	0.855235
$OvR - AUC - weighted$	0.854823	**0.862879**	0.854823

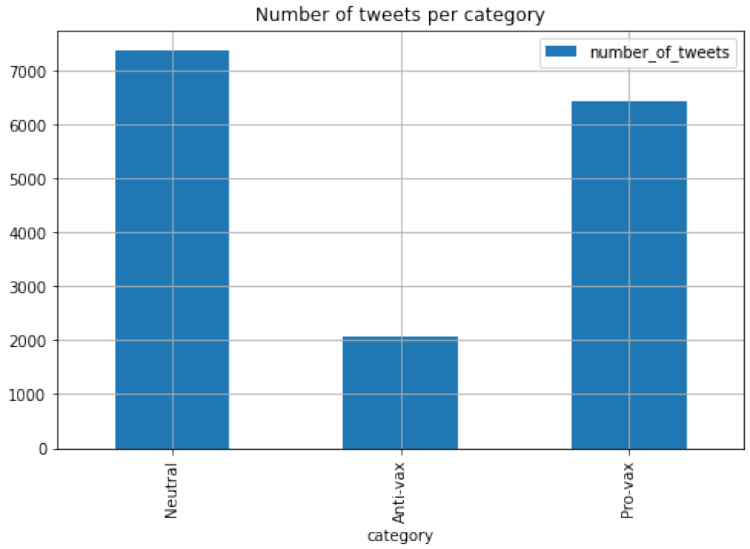
One-vs-One and One-vs-Rest are methods for using binary classification algorithms for multi-class classification. One-vs-Rest splits the dataset into one binary dataset for each class, while One-vs-One approach splits the dataset into one dataset for each class versus every other class. The results are satisfying.Plus, I used the mean absolute error metric, even though it is probably not the best and most trustworthy metric for a multi-class classification problem:

	$CountVec$	$HashingVec$	$TfidfVec$
$MAE - Training$	0.3743	**0.3083**	0.3743
$MAE - Test$	0.4421	**0.4294**	0.4421

The Hasing Vectorizer seems again to get better results, as the error is the minimum out of the three vectorizers.

2.5 Plots

In general, in this study, I wanted to experiment as much as possible with Data Visualization, since it is an essential part of Machine Learning. First of all, I checked the balance of the dataset.



I have added some plots where the most important words are presented, where we can see the expected words (e.g. covid,pandemic,death,vaccine,jab or numbers that must indicate dates or number of cases).In this way, also, I could make sure that I use efficiently the Vectorizer and the TfIdfTransformer.

Also, I plotted the learning curves, along with a performance and scalability curve, to ensure that the model is not overfitting or underfitting and that it performs well, when the instances increase. The training and validation score curves converge and have the expected behavior, when it comes to the first and the third models. However, the Hashing Vectorizer model, despite its efficient results, seems to be overfitting. The conclusion I drew, after trying out a plethora of different configurations,is that without all these hyperparameters that I added to the Hashing Vectorizer, the model does not overfit significantly, but the results are worse than those of the other models. So, in any case, the other two models are preferable and probably the CountVectorizer one is the best, only for the reason that it is faster than the TfIdfVectorizer.

2.6 Conclusion

In this assignment, I had the opportunity to familiarize with the concepts of data preprocessing, extraction, vectorization, visualization, but also with the process of

making plenty of tests and research, in order to achieve the best result. Finally, I faced the challenge of overfitting and I ended up with a decent model, taking into account the relatively small and imbalanced dataset that I used. However, there is much room for improvement.

References

[1] NLTK Documentation

[2] Scikit-Learn Documentation

[3] Vectorization

[4] Word Cloud