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 Logistic Regression classifier, which classifies tweets as Neutral, Pro-vax or Anti-vax.

2.1 Pre-processing

First of all, I performed some pre-processing 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, Lancaster, Snowball), but I noticed that the Wordnet Lemmatizer performed much better than all of them, so I did not include these tests in the notebook at all. In general, what I understood by my research is that lemmatization provides better results at times, because it performs an analysis that depends on the word's part-of-speech 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 test data:

	Porter	*Lancaster*	*Snowball*	*Lemmatizer*
Recall	0.8203	0.8129	0.8194	**0.8376**
Precision	0.824	0.816	0.8231	**0.8417**
$F1$	0.8136	0.8068	0.8127	**0.832**
Accuracy	0.8203	0.8129	0.8194	**0.8376**

	Porter	*Lancaster*	*Snowball*	*Lemmatizer*
Recall	0.7191	0.7142	0.7212	**0.7243**
Precision	0.7156	0.7145	0.7189	**0.7205**
$F1$	0.7108	0.7080	0.7135	**0.7175**
Accuracy	0.7191	0.7142	0.7212	**0.7243**

2.2 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, HashingVectorizer with TfidfTransformer and TfidfVectorizer. The HashingVectorizer did not perform as well as the others and TfidfVectorizer had slightly better results in the testing, while CountVectorizer has better results in the training.

The results on both training and testing data:

	$CountVec$	$HashingVec$	$TfidfVec$
Recall	**0.8376**	0.8282	0.8371
Precision	**0.8417**	0.8322	0.8410
$F1$	**0.8320**	0.8224	0.8317
Accuracy	**0.8376**	0.8282	0.8371

	$CountVec$	$HashingVec$	$TfidfVec$
$Recall$	0.7243	0.7169	**0.7265**
$Precision$	0.7205	0.7162	**0.7262**
$F1$	0.7175	0.7083	**0.7203**
$Accuracy$	0.7243	0.7169	**0.7265**

2.3 Scaling

I would also like to note that I experimented with MinMaxScaler and StandardScaler with Logistic Regression, and even though the training accuracy was very high (greater than 0.9), the test accuracy was lower than without scaling, so I assume that it is overfitting and I removed the scaler.

Finally, using different metrics, precision, recall, f1, accuracy, confusion matrix, I was able to see the quality of my model.

2.4 Plots

I have added some plot where the most important/rare words are presented. Also, I have a learning curve plot, where we can see that the model is not overfitting.

2.5 Classifiers

Finally, I experimented with some other classifiers as well, in order to check whether the results will be any

improved, in comparison with multinomial Logistic Regression.

The results are the following, using TfidfVectorizer for both training and test data:

	$DecisionTree$	MLP	$BernoulliNB$
$Recall$	0.9937	0.9935	0.7883
$Precision$	0.9937	0.9935	0.8
$F1$	0.9937	0.9935	0.7779
$Accuracy$	0.9937	0.9935	0.7883

	$DecisionTree$	MLP	$BernoulliNB$
$Recall$	0.6599	0.6148	0.695
$Precision$	0.6581	0.6201	0.7087
$F1$	0.658	0.6168	0.6807
$Accuracy$	0.6599	0.6148	0.695

They did not perform well and I assume that there is a good chance they are overfitting.