Homework 1

Βησσαρίων Μουτάφης sdi1800119@di.uoa.gr
1115201800119

1. MSE Derivative

First lets prove $\frac{\partial}{\partial w_j} MSE(\boldsymbol{w}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{w} \cdot \boldsymbol{x^{(i)}} - y^{(i)}) x_j^{(i)}$.

We know that

$$\boldsymbol{w} \cdot \boldsymbol{x} = \sum_{j=1}^{n} w_j x_j \Rightarrow \frac{\partial}{\partial w_j}(\boldsymbol{w} \cdot \boldsymbol{x}) = \frac{\partial}{\partial w_j}(\sum_{j=1}^{n} w_j x_j) = x_j$$

so, we can conclude that

$$\frac{\partial}{\partial w_j} MSE(\boldsymbol{w}) = \frac{1}{m} \cdot \frac{\partial}{\partial w_j} \sum_{i=1}^{m} (\sum_{j=1}^{n} w_j x_j - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} (\sum_{j=1}^{n} w_j x_j - y^{(i)})^2$$

that will ultimately leave us with

$$\frac{\partial}{\partial w_j} MSE(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} 2 \cdot (\sum_{j=1}^{n} w_j x_j - y^{(i)})(\frac{\partial}{\partial w_j}(\sum_{j=1}^{n} w_j x_j - y^{(i)})) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{w} \cdot \boldsymbol{x^{(i)}} - y^{(i)}) x_j^{(i)}$$

Now we can proceed in the actual proof. We begin with the derivative of our MSE function:

$$\nabla MSE(\boldsymbol{w}) = \begin{pmatrix} \frac{\partial}{\partial w_1} MSE(\boldsymbol{w}) \\ \frac{\partial}{\partial w_2} MSE(\boldsymbol{w}) \\ . \\ . \\ . \\ \frac{\partial}{\partial w_n} MSE(\boldsymbol{w}) \end{pmatrix}$$

We know that for each one of the partial derivatives we can write

$$\frac{\partial}{\partial w_j} MSE(\boldsymbol{w}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{w} \cdot \boldsymbol{x^{(i)}} - y^{(i)}) x_j^{(i)} = \frac{2}{m} \boldsymbol{X_j} (\boldsymbol{X} \cdot \boldsymbol{w} - \boldsymbol{y})$$

where $X_j \in R^{1 \times m}$ and

$$X_j = \begin{pmatrix} x_j^{(1)} & x_j^{(2)} & ... & x_j^{(m)} \end{pmatrix}$$

.

We now can re-write the derivative of MSE as

$$\nabla MSE(w) = \begin{pmatrix} \frac{2}{m}\boldsymbol{X_1}(\boldsymbol{X}\cdot\boldsymbol{w}-\boldsymbol{y}) \\ \frac{2}{m}\boldsymbol{X_2}(\boldsymbol{X}\cdot\boldsymbol{w}-\boldsymbol{y}) \\ . \\ . \\ . \\ \frac{2}{m}\boldsymbol{X_n}(\boldsymbol{X}\cdot\boldsymbol{w}-\boldsymbol{y}) \end{pmatrix}$$

that can be reformed to

$$\nabla MSE(w) = \begin{pmatrix} \boldsymbol{X_1} \\ \boldsymbol{X_2} \\ . \\ . \\ . \\ \boldsymbol{X_n} \end{pmatrix} \cdot \frac{2}{m}(\boldsymbol{X}\cdot\boldsymbol{w}-\boldsymbol{y})$$

but we know that $\begin{pmatrix} \boldsymbol{X_1} \\ \boldsymbol{X_2} \\ . \\ . \\ . \\ \boldsymbol{X_n} \end{pmatrix} = \boldsymbol{X}^T$ so we conclude that

$$\nabla MSE(w) = \frac{2}{m}\boldsymbol{X}^T\cdot(\boldsymbol{X}\cdot\boldsymbol{w}-\boldsymbol{y})$$

2. Sentiment Analysis

In this section we will talk about the sentiment analysis we conducted onto the given datasets. We used the following pipeline in order to properly extract the features from the tweets:

1. Cleaned the Tweets (check following section)
2. Lemmatized the Tweets
3. Constructed and fitted a vectorizer (3 different kinds to be exact) and transformed the text array into a feature matrix
4. Applied PCA dimensionality reduction algorithm in order to keep only the most important features and avoid overfitting of the Softmax regression.

2.1 Data Cleaning

We applied the following text cleaning pipeline:

1. Transform all the words to the lowercase equivalent.

2. Remove *http://* tag since it's useless.

3. Divide the URL parts when there is one since there might be a correlation between domains and anti/pro vax tweets.

4. Remove any stopwords, except some of the stopwords that provide sentiment. such as words that show strong opinions in negative/positive manner. They might be highly related with the sentiment of the tweet.

5. Remove any punctuation.

In this way we make sure that sentences are consisted only by important words and that they are kind of "normalized" in order to be distinguished among the different ways they be addressed.

2.2 Lemmatization

Another important part of text pre-processing was the lemmatization of all sentences in order to decrease the feature dimensionality and group all the relative words among the tweets in order to make a more throughout sentiment analysis. You can see that we used an *nltk* lemmatizer and that we transformed all lemmas to their respective verb *part of speech*.

2.3 Vectorizing

We try vectorizing the **cleaned** tweets in 3 different ways

1. CountVectorizer: Common Bug of words implementation

2. HashVectorizer: Another version of bug of words

3. TF-IDF Vectorizer: term frequency-inverse document frequency to measure how relevant is a word to the tweets theme

Each one of those vectorizers gave a different perspective on the features and with proper tuning, we managed to extract most features.
To see how well each one of those feature extractors performed check the Evaluation section.

2.4 PCA Dimesionality Reduction

We used the principal component analysis, in order to keen n most important text features so that we prevent softmax-model from over-fitting training data.

2.5 Training

We created 3 different models, one for each vectorizer. After we normalized training data with a *StandardScaler*, we trained the multinomial logistic regression models from *sklearn's* library. At this points we are ready to present the evaluation metrics over the test-set, we were given.

2.6 Evaluation

Evaluation process was undergone using the following metrics:

- *Precision* in order to evaluate model's classification ability to classify correctly the retrieved instances given a class.
- *Recall* in order to evaluate the model's classification ability to retrieve correctly relevant instances for each class.
- *F1-Score*, which gives a grasp of the generalization accuracy
- *Accuracy* as a general metric of how well the model can predict the class of a given object that has never seen before.
- *Log Loss* since we are interested in monitoring how well the model could generalize on un-seen data.

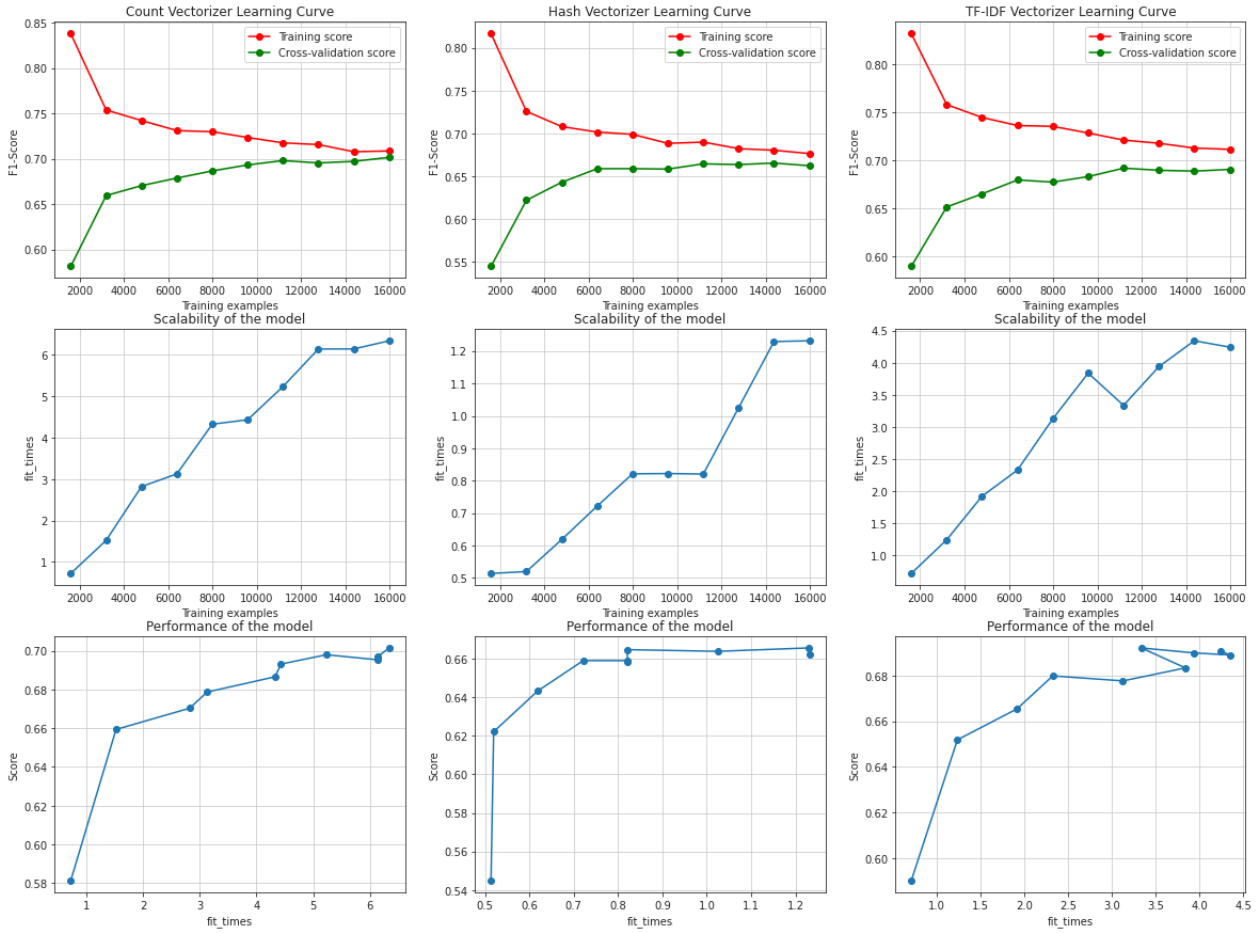
In our evaluation process we used **weighted** precision, recall, f-score and loss since we wanted to eliminate any abnormal results because of label imbalance. To watch test-set performance you could open the Jupyter notebook in section **Softmax Evaluation**, where we present evaluation results for all 3 models.

2.7 Learning Curves and Final Notes

Finally, we present the learning curves (see below), that we constructed by using sklearn's *learning_curve* function.

We had to pass all (test and training) data in the function, so we had to concatenate the two beforehand. In order to evaluate only on given evaluation set we supplied the function with a *cv* arguments that is consisted of indices on the range of the test data. We also supplied the user with an easy K fold function that will train the model with the same 16k of training data but will evaluate with a different slice of test data, whether the reader wants to monitor models behaviour in a more general manner than 1 fold evaluation.

After this we used a made-up function (check *References*) to print the results:



From these graphs we came to the following conclusions:

- Best f1-score came from the Count and TF-IDF vectorizers so they are the best feature extractors given our model predictive power.
- *Training-Evaluation* scores are converging in all 3 cases which means that our models **neither overfit nor underfit**. This means that the general **predictive power** of Classic multinomial logistic regression (which is a linear boundary classifier) is quite **low** given the classification task at hand.
- All models seems to need a fair amount of iterations to converge as we can see from the scalability graphs and that means they fairly scalable, especially in the CountVectorizer case.
- We can also watch that the evaluation performance is converging to its peak really fast so that means that the model reaches its score threshold relatively fast. Again the best performance is met in the CountVectorizer model.
- The models are all very good fitted but the problem of sole linear explanatory power in the model's prediction function, is setting some constraints and thresholds in terms

of predictive accuracy. So we conclude that any neural network with simple non-linearities could score a higher accuracy with same preprocessed datasets.

2.8 References

- *sklearn's Learning Curve Graphs*
- sklearn metrics
- https://raw.githubusercontent.com/jacobeisenstein/gt-nlp-class/master/notes/eisenstein-nlp-notes.pdfEinstein NLP Notes