Τεχνητή Νοημοσύνη II - Εργασία 1

Μυστακίδης Ιωάννης 1115201600113

1.

Our hypothesis function h_w is:

$$h_w(x) = w_0 + w_1 x_1 + ... + w_n x_n$$

which in vectorized form is:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

The MSE is:

$$MSE(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2$$

Now, taking the partial derivative of MSE with respect to the bias term w_0 we get:

$$\begin{aligned}
\frac{\partial}{\partial w_0} MSE(\mathbf{w}) &= \frac{\partial}{\partial w_0}(\frac{1}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2) \\
&= \frac{1}{m} \frac{\partial}{\partial w_0} \sum_{i=1}^{m} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2 && \text{(By the Scalar Multiple Rule)} \\
&= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_0} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2 && \text{(By the Sum Rule)} \\
&= \frac{1}{m} \sum_{i=1}^{m} 2(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \frac{\partial}{\partial w_0} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) && \text{(By the Power Rule and Chain Rule)} \\
&= \frac{2}{m} \sum_{i=1}^{m} (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})
\end{aligned}$$

where at the last step:

$$\frac{\partial}{\partial w_0}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) = \frac{\partial}{\partial w_0}(w_0 + w_1 x_1^{(i)} + ... + w_n x_n^{(i)} - y^{(i)}) = 1$$

because all $x's$, $y's$ are constants and also, since we're taking the partial derivative with respect to w_0, all $w's$ are constants too except w_0, whose derivative equals 1.

Similarly, taking the partial derivative of MSE with respect to any term w_j, except the bias term w_0, we get the same result except in the last step where:

$$\frac{\partial}{\partial w_j}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) = \frac{\partial}{\partial w_j}(w_0 + w_1 x_1^{(i)} + ... + w_n x_n^{(i)} - y^{(i)}) = x_j^{(i)}$$

because again all $x's$ and $y's$ and now the only non-constant w term is w_j whose derivative will be 1 multiplied by the constant $x_j^{(i)}$ (all other terms end up 0 because they're constants).

Therefore:

$$\frac{\partial}{\partial w_j} MSE(\mathbf{w}) = \frac{2}{m}\sum_{i=1}^{m}(h_{\mathbf{w}}(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Now, starting backwards from what we want to prove:

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

We know $\mathbf{w}$ is:

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

and $\mathbf{X}$ is:

$$\mathbf{X} = \begin{pmatrix} (\mathbf{x}^{(1)})^T \\ (\mathbf{x}^{(2)})^T \\ \vdots \\ (\mathbf{x}^{(m)})^T \end{pmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

therefore $\mathbf{X}\mathbf{w}$ is:

$$\mathbf{X}\mathbf{w} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{bmatrix} x_0^{(1)}w_0 & + & x_1^{(1)}w_1 & + & x_2^{(1)}w_3 & + & \cdots & + & x_n^{(1)}w_n \\ x_0^{(2)}w_0 & + & x_1^{(2)}w_1 & + & x_2^{(2)}w_3 & + & \cdots & + & x_n^{(2)}w_n \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots \\ x_0^{(m)}w_0 & + & x_1^{(m)}w_1 & + & x_2^{(m)}w_3 & + & \cdots & + & x_n^{(m)}w_n \end{bmatrix}$$

hence $\mathbf{X}\mathbf{w} - \mathbf{y}$ is:

$$\mathbf{X}\mathbf{w} - \mathbf{y} = \begin{bmatrix} x_0^{(1)}w_0 & + & x_1^{(1)}w_1 & + & x_2^{(1)}w_3 & + & \cdots & + & x_n^{(1)}w_n - y^{(1)} \\ x_0^{(2)}w_0 & + & x_1^{(2)}w_1 & + & x_2^{(2)}w_3 & + & \cdots & + & x_n^{(2)}w_n - y^{(2)} \\ \vdots & & \vdots & & \vdots & & \ddots & & \vdots \\ x_0^{(m)}w_0 & + & x_1^{(m)}w_1 & + & x_2^{(m)}w_3 & + & \cdots & + & x_n^{(m)}w_n - y^{(m)} \end{bmatrix} = \begin{bmatrix} h_{\mathbf{w}}(x^{(1)}) & - & y^{(1)} \\ h_{\mathbf{w}}(x^{(2)}) & - & y^{(2)} \\ \vdots & & \vdots \\ h_{\mathbf{w}}(x^{(m)}) & - & y^{(m)} \end{bmatrix}$$

Now, $\mathbf{X}^T$ is:

$$\mathbf{X}^T = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & x_0^{(3)} & \cdots & x_0^{(m)} \\ x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} & \cdots & x_n^{(m)} \end{bmatrix}$$

and, therefore, $\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$ is:

$$\begin{aligned}\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) &= \begin{bmatrix} x_0^{(1)} & x_0^{(2)} & x_0^{(3)} & \cdots & x_0^{(m)} \\ x_1^{(2)} & x_1^{(2)} & x_1^{(3)} & \cdots & x_1^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} & \cdots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} h_{\mathbf{w}}(x^{(1)}) & - & y^{(1)} \\ h_{\mathbf{w}}(x^{(2)}) & - & y^{(2)} \\ \vdots & & \vdots \\ h_{\mathbf{m}}(x^{(m)}) & - & y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} (h_{\mathbf{w}}(x^{(1)}) - y^{(1)})x_0^{(1)} & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)})x_0^{(2)} & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)})x_0^{(m)} \\ (h_{\mathbf{w}}(x^{(1)}) - y^{(1)})x_1^{(1)} & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)})x_1^{(2)} & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)})x_1^{(m)} \\ \vdots & & \vdots & & \vdots & & \ddots \\ (h_{\mathbf{w}}(x^{(1)}) - y^{(1)})x_n^{(1)} & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)})x_n^{(2)} & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)})x_n^{(m)} \end{bmatrix}\end{aligned}$$

and because x_0 is always equal to 1 (for all m observations):

$$\begin{aligned}\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) &= \begin{bmatrix} (h_{\mathbf{w}}(x^{(1)}) - y^{(1)}) & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)}) & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)}) \\ (h_{\mathbf{w}}(x^{(1)}) - y^{(1)})x_1^{(1)} & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)})x_1^{(2)} & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)})x_1^{(m)} \\ \vdots & & \vdots & & \vdots & & \ddots \\ (h_{\mathbf{w}}(x^{(1)}) - y^{(1)})x_n^{(1)} & + & (h_{\mathbf{w}}(x^{(2)}) - y^{(2)})x_n^{(2)} & + & \cdots & + & (h_{\mathbf{m}}(x^{(m)}) - y^{(m)})x_n^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \\ \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})x_1^{(i)} \\ \vdots \\ \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})x_n^{(i)} \end{bmatrix}\end{aligned}$$

and, finally, $\frac{2}{m}(\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}))$:

$$\frac{2}{m}(\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})) = \begin{bmatrix} \frac{2}{m}\sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \\ \frac{2}{m}\sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})x_1^{(i)} \\ \vdots \\ \frac{2}{m}\sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})x_n^{(i)} \end{bmatrix} = \begin{pmatrix} \frac{\partial}{\partial w_0} MSE(\mathbf{w}) \\ \frac{\partial}{\partial w_1} MSE(\mathbf{w}) \\ \vdots \\ \frac{\partial}{\partial w_n} MSE(\mathbf{w}) \end{pmatrix} = \nabla_{\mathbf{w}} MSE(\mathbf{w})$$

which is what we were expecting to get as we showed above when taking the partial derivatives with respect to the w terms.

2.

How to Run:

- Firstly, you have to upload the datasets on colab. We read the training and validation data in the first cell. The paths of the datasets are simply the names of the datasets themselves, but you can change according to the path on which you upload the files yourself.
- You can either:
  - Use 'Run all'

    Which would take a little more than 1 hour to finish (because of all the learning curves plotting and the fine-tuning). We did exploit some of the (cpu-level) parallelization capabilities of sklearn using the n_jobs parameter where applicable, but google colab provides only 2 cpu cores compared to 4 on kaggle and 6 on our local machine. So did try out way more experiments (especially concerning the data fine-tuning explained below) that are not shown on this particular notebook.
  - Or omit some (code) cells and run the others:

    You can omit the cells that plot the learning curves in the 'Model Selection and Training'. That is:
    * Cell 11: which plots the validation curves for the C regularization parameter of LogisticRegression.
    * Cell 12: which plots the learning curves for various values of the ngram_range parameter of the vectorizer.
    * Cell 13: which plots the validation curves for various values of the min_df parameter of the vectorizer.
    * Cell 14: which plots the learning curves for various values of the strategy parameter of the vectorizer.

    These cells are merely for visualization purposes only, just to justify our choices for the various hyperparameters (although countless more experiments did take place that are not shown). They can be omitted when running the notebook because their results are already plotted and they're not going to change if you re-run them because we've set the random seed to 42 in the beginning for reproducible results. Omitting these cells will reduce the running time about half an hour.
    You can't omit any of the other cells because they are important for the flow of the notebook (for example, after fine-tuning the preparation steps parameters we use the output of GridSearch in the next cells etc.).

- Finally, in order to run the model on the test set you only have to change the path of the fine in the section 'Final Model Evaluation on the Test Set' (the path is currently set to the validation set again and there are some prints and plots). Then you can run the cells in that section only to see how our model performs on the test set.

Notebook Sections:

- Data Loading and Quick Look at the Structure

  Where we simply load the training and validation data and take a quick look at the structure of the training data (we don't snoop at the validation data).

- Data Exploration

  Where we explore the training set more carefully. We find and plot the most popular words (unigrams) in the different sentiment categories. We also find and count some possible noise in our data like links and emojis.

- Data Preparation

  This is the data preprocessing part. We clean and vectorize the data. We clean the noise we found in the previous step, like links, hashtags etc. Then we normalize the data by converting to lowercase, removing punctuation etc and we finally vectorize our data using either plain Bag of Words or TF-IDF.

- Model Selection and Training

  Here we explore how the different parameters of some of the preprocessing steps and the classification algorithm itself (e.g. the C regularization parameter of LogisticRegression or the min_df parameter of the vectorizer) affect the bias and variance of our model and we present various plots (learning and validation curves) to visualize this.

- Error Analysis

  In this section, we perform some error analysis on what we believe is the (so far) best model (best preparation steps and model parameters) according to the experimentation that took place in the previous step. We print the classification report, the confusion matrix, some examples of wrong classification and the top n-grams per class found by our model.

- Fine-Tuning

  Where we fine-tune the (quite sufficient) model that we performed error analysis on earlier.

  - We perform some **data** fine-tuning (using GridSearch) by checking whether some of the other preprocessing steps that we didn't experiment with during 'Model Selection and Training' (e.g. whether to completely remove emojis or convert them to their text representation instead, choosing between Stemming and Lemmatization, removing/keeping stopwords etc.) can improve our predictions on the validation set. Then we plot the learning curve again to show that model still doesn't overfit/underfit the training data.
  - Then, we perform some **model** fine-tuning (using RandomizedSearch) by checking whether a (only slightly) different value for the C regularization parameter of LogisticRegression can improve our predictions with hurting our generalization capabilities (therefore, we only search for the best C on a narrow range that we already showed generalizes well in 'Model Selection and Training').

- Final Model Evaluation on the Test Set

  Finally, after fine-tuning, we have the best preparation pipeline from GridSearch (best_preparation_pipeline) and the best model from RandomizedSearch (final_model), so we simply read the test data (this is where you change the path of the data), transform the test data (no fit) using the best preparation pipeline and then predict on the test data using the best model (final_model). Again, we print the classification report, the confusion matrix, some examples of wrong classification and the top n-grams.

Insights:

- Data Exploration:

  When we plotted most common words per each class, the results looked quite similar. As explained in the notebook, it appears that unigrams alone are not very informative and helpful for the prediction of the label. It also appeared that there is a lot of noise, like links, in our data. We then showed the number of links, emojis, tags and hashtags contained in our tweets and we made functions for removing these. We treat these preparation steps as hyperparameters because we don't know how well they work beforehand. For example, is it better to completely remove emojis or convert them to their text representation? Is it better to remove hashtags completely or leave them and only remove the '#' symbol (when removing punctuation), because some hashtags convey emotion like #Awesome?

- Model Selection and Training:

  - C Regularization Parameter:

    The validation curve we plotted shows us that the optimal value lies somewhere between 0.05 and 0.15. A value less than 0.05 constraints our model too much and probably leads to some underfitting on the training data, whereas a value higher than 0.15 makes our model overfit the trainning data and hurts its generalization capabailities.

  - ngram_range Parameter of the Vectorizer:

    Using more than unigrams allows us to fit the training set better, but using way too many of them (e.g. up to five-grams) without any way to constraint the number of features produced (like the min_df parameter explained next) leads to high variance because of overfitting. Exlcuding unigrams leads to lower variance but higher bias, so it seems like we're kind of underfitting the training data this way.

  - min_df Parameter of the Vectorizer:

    This parameter helps constrain the number of features extracted by our vectorizer, because it sets a minimum number of instances that a word must be present in to be considered a feature. It seems that its better to use a value greater than 1 (which considers all words as features no matter how often they appear in the data) but also not a very high value, because even though this higher value leads to smaller variance, it also leads to higher bias and it looks like we could do better than that.

  - Vectorizer Strategy:

    It looks like the plain Bag of Words method leads to better f1_scores (lower bias) but a little larger variance than TF-IDF. Choosing between the two depends on whether we consider the variance in Bag of Words significant enough to go with the lower variance TF-IDF model.

  We treat the other preparation steps as hyperparameters when fine-tuning the model. It's not worth to mention exactly how they affect the model because they don't make that much of a difference to the generalization of our model and only slightly affect the f1_scores. There are some more insights and more commenting on the results mentioned in the notebook but not here because we'd just be repeating the same stuff twice.

Possible Additional Experiments:

- In the preparation steps we could have also tried:

  - Removing emoticons (they differ from emojis)

  - Removing all numbers or convert them to their text representation
  - Removing all non-ascii characters (because it appears that there are some chinese characters present in the data, most of which where probably not considered as features though because of the min_df parameter of the vectorizer)
  - Removing the most frequent and/or the most rare words
  - Converting abbreviations to their full equivalents (e.g. 'a.k.a.' to 'also known as')
  - Perform some spelling correction
  - Use a sentiment lexicon (e.g. affin) to create our own vocabulary for the vectorizer only with words conveying sentiment, or check which hashtags are sentiment words and which are not and only strip the non-useful ones and many more things using the valence of each word.

  Again, we could treat these as hyperparameters and see which combination of preparation steps would be best.

- In the model training we could have also tried:
  - Adding the max_features parameter on the vectorizer to see if it could improve the predictions and see how it affects the generalization.
  - Experimented more with the tol hyperparameter of LogisticRegression, apart from the default 0.0001 and 0.001.
  - We did experiment with other values for the penalty hyperparameter of LogisticRegression (like l2 and elasticnet with l1_ratio=0.9) which is not shown in the notebook, but it appears that l1 worked best.