Exercise 3 - Language Identification with CNNs

This project is about language identification using convolutional neural networks (CNNs). In a first step, the most important distributional properties of the dataset are inspected and plotted. The dataset contains a total of 65954 samples split into 52675 training samples and 13279 test samples. Every sample comprises a "tweet" column of the textual content and a "label" column of the associated language. The histogram shows that there are many labels with very low frequency (strong class imbalance). For example 60 lanuages occur less than 100 times in the whole dataset while English occurs more than 20,000 times. We are dealing with this class imbalance in a later step. We merge the training and the test data into one dataframe for easier pre-processing and to ensure that training and test data comes from the same distribution. We saw that some languages/ labels do not appear in the training data while appearing in the test data and the other way around Since it is not required that we merge training and test data into one dataset and before starting to train, we randomly shuffle and split into train and test data again. This ensures that all labels appear in the training and test data.

Pre-Processing

We decided to use a very similar preprocessing process similar to project number 1. First of all we remove patterns like retwees, hyperlinks, emojis, xml, hashtags which do not help with the classification. Then we detect all languages with a small number of occurences (< 100). We use back translation and upsamling in order to increase the sample size to at least 100. The labels are then transformed into indices and the tweets are tokenized and vectorized and then padded to all have a unit length of 80. Next, we split the vectorized data into training and test data based on a 80:20 split. 10 percent of the training data is then used as validation data.

Part 1

We decided to build the CNNs using keras. We tried out different combinations of optimizer, learning rate, dropout, number of filters, stride, kernel size, different pooling strategies and batch sizes. Since we are not dropping any classes from neither the training nor the train set we see that our best classifier performs very well on average (Weighted F1 score of 97%). However, if you consider the macro F1 score (which weights the F1 equally for every label) you can see that the model performs not really well, because it only achieves a macro F1 of 50%. An easy trick in order to increase the macro F1 score, would be to drop all the labels with small sample size from the training and test set.

Model Nr.	Optimzer	Learning Rate	Dropout Rate	#Filters	Stride	Kernel Size	Batch Size	Accuracy	Macro F1	Weighted F1
1.	sgd	0.1	0.5	128	1	3	64	0.84	0.19	0.83
2.	sgd	0.01	0.25	256	1	7	128	0.89	0.15	0.88
3.	adam	0.01	0.25	512	2	3	64	0.88	0.15	0.89
4.	adam	0.001	0.5	128	1	5	64	0.95	0.5	0.94
5.	adam	0.001	0.25	128	1	5	128	0.97	0.5	0.97

Part 2

Take the best performing model and evaluate it on the test set. We decided to evaluate model number 5 on the test data since it showed the best validation accuracy. The best models performance and the used hyperparameters can be seen in the following table.

Model Nr.	Optimzer	Learning Rate	Dropout Rate	#Filters	Stride	Kernel Size	Batch Size	Accuracy	Macro F1	Weighted F1
5.	adam	0.001	0.2	128	1	5	128	0.92	0.5	0.91

Part 3

Reasons about the observed effects of your 5 best hyperparameter settings on model performance..
-1. Optimizer => We saw that adam generally performed better on our training data leading to better results than sgd. The reason is probably that standard sgd with no momentum descends less efficiently and it is easier to get trapped in local miminmum while adam with adaptive descending speeding and momentum usually can have better optimization performance in gradient descent.
-2. Learning rate => If the learning rate was to high (e.g 0.1, 0.01) we saw that the training process did not really converge. So we used low learning rate in our model 5. To prevent the loss function from not converging, we can also try learning rate decay in the future.
-3. Dropout => Very big values for the dropout rate (e.g 0.8) lead to very poor results. Since dropout is firstly intend to solve overfiting, as we set a over high dropout rate, the model became underfitting. After many tryouts, we found that the optimal dropout rate of our model seems to be located in between 0.25 and 0.5.
-4. Number of filters => We were able to see that increasing the number of filter did not improve the performance at some point anymore. In our case this was around 128.
-5. Strides => A strid of one seems optimal, shifting the filter step by step over the input.
-6. Different kernel sizes => The optimal size for the kernel seems to be five, while 3 was to small and 7 was to big. This seems to be related to our tokenization. Since we tokenized our data on a character level, each inputs of the model reprecents one character. Suppose we have a kernel of size 3, As the kernels scan through the input, every stride it reads 3 characters, which does not seems to enough to distinguish many languages.
-7. Pooling strategies => We only used GlobalMaxPooling as the Pooling strategy.
-8. Batch sizes => Reasonable batch sizes, between 32, 64 and 128 all seemed to work reasonably well.

Part 4

Compare the outputs of the best CNN model to your best performing model from Exercise 1. Which classifier scores higher on the test set? Do you have an idea, why this might be?

Model	Accuracy	Macro-F1	Weighted-F1
SGDClassifier	0.93	0.93	0.93
MultinomialNB	0.86	0.84	0.86
ANN	0.92	0.89	0.92
CNN	0.92	0.5	0.91

Multiple causes might be the reason while the models from assignment 01 perform better (especially the SGDClassifier) than the CNN we used now. First of all, we were allowed to optimize the hyperparameters on the training data for assignment 01. Furthermore, for assignment 03 we upsampled the underrepresented classes to a total occurence of 100. Therefore in the split we are using for training and testing more samples from a formerly underrepresented class occur. However, 100 is still small compare to large classes English's 20,000. Since the model performs worse for samples from small classes, this reduces our macro f1 score. Overall, the weighted average performance of the models is still very comparable.