Cojocariu Sebastian
Group 407

Project 1

For this project I used the following methods:

- CharCNN + WordCNN (regression) – Winner
- Kmeans + CharCNN + WordCNN (classification)
- NuSVR using String Kernel

The preprocessing step

- Each document was first converted to lower-case (in order for the word embedding to be consistent since we didn't want words like "isch" and "Isch" to have their own embedding/ be considered unrelated by the classifier).
- Then, each punctuation sign was replaced by a blank space (as I considered punctuation as a minor factor for this task). This step was not always executed (as I tested with both removing and non-removing the punctuation).
- Then one can choose to remove non alpha numerical characters or keep them as independent words (keep only [a-zA-ZÀ-ž0-9] characters or all the them, such as emojis or special unicode) (*customizable* during training).
- Trailing spaces were then removed.
- After that, the new document is tokenized and have all the stop words removed (german). Words containing both numerical and letters are removed as well (as they were suspected to provide no real information).
- *Various combination of these steps* were used during the experiments.

Models

1,2) CharCNN + WordCNN architecture (for both classification and regression task)

The architecture consists of two parts (which can be discarded during training):

- A character-level CNN
- A word-level CNN

The first part is described perfectly here: https://arxiv.org/pdf/1509.01626.pdf . The *vocabulary of characters* is selected based on the frequency of these characters (it can contain also non-alphanumerical characters such as the most frequent emojis as they may be useful to distinguish between different regions). The threshold, as well as the removal/keep of the non-alphanumerical characters are *customizable.*

The idea to consider the word-level CNN arise from the fact that maybe position of multiple words combined can be relevant to create useful features (the order of words can be important in some regions). In order to reduce the dimension of the *vocabulary*, each word was mapped to its stemmed conversion in the dictionary (meaning that the dictionary contains only *stemmed* words as keys). Also, for a word to be considered part of the vocabulary, the number of its occurrences must be bigger than a minimum threshold (which is again *customizable*).

Words Embedding

First, I tried Bert for word embeddings (german bert embedding available https://deepset.ai/german-bert or as "bert-base-german-cased" in TFBertModel). I used the last 4 hidden_states (averaging them as I read on some articles it is the most robust way to get embeddings) => tensors of size 768 for each word.
The results were quite poor with this approach, with a MAE in the range of 0.75-0.8.

Cojocariu Sebastian
Group 407

Since Bert pretrained embedding didn't work as expected, I started training my own embeddings as part of the deep learning model
An important observation is that Google Translate doesn't proper translate the tweets provided, so maybe they are o combination of multiple dialects from Germany or Switzerland.

Classification Method

First, I tried to transform this regression problem into a classification one (using KMeans to cluster the regions (as I saw a good clustering when plotting the data) and then train a classifier to assign one region to each sample). Using these five clusters, if I could achieve a perfect assignation of the regions using the deep learning model, MAE would have been 0.36. Unfortunately, the classifier overfits very fast (even increasing the dropout to 0.5 and using spatialDropout + GaussianNoise + Adam with weight decay), with almost 100% accuracy on the training set and just about 68 % accuracy on the validation set (using 5 regions). I tried training a smaller model without a very consistent improvement, so I switch to solving this problem as a regression one instead.

The architecture consists of:

- **Character CNN** submodule: 3 layers of convolutions followed by a flatten layer and a dense layer (easy *customizable* in the model.py).
- **Word CNN** submodule: 3 layers of convolutions followed by a flatten layer and a dense layer (easy *customizable* in the model.py).
- The selection of training the charCNN or wordCNN submodule is *customizable*.
- Concatenation of the outputs of each final dense.
- Some fully connected layers with different nonlinear activations (ReLU, Sigmoid) to solve the task.

In order to avoid overfitting, several *regularization techniques* were used:

- **Dropout** after each Dense/Linear layer: the most suitable one is in the range 0.03-0.1 (smaller values creates an overfitting regime, while larger ones make the model doesn't train at all).
- **SpatialDropout** aplied immediately after each convolutional layer: in the range of 0.005 – 0.02.
- **GaussianNoise** applied immediately after each convolutional layer: in the range of 0.001 – 0.005.
- **Weight Decay** for **Adam Optimizer** in the range of 10^-6 – 10^-7.

The Learning Rate:

- Selected in the range 10^-2 - 10^-4. Best value was the default one 10^-3
- After each epoch the learning rate is multiplied by a constant factor, usually between 0.997-0.999. The constant factor and the initial learning rate are *customizable.*

The Loss Function:

- *Mean Absolute Error (MAE):* the problem with it is that the derivate is constant no matter the error.
- *Mean Squared Error (MSE):* the problem is that it penalizes errors larger than 1 too much.
- *Huber Loss:* a combination of both (being more robust to large errors).
- The loss function is *customizable.*

Among all these losses, the best results were achieved using **Huber Loss.**

Cojocariu Sebastian
Group 407

Hyperparameters:

- The *maximum number of characters* per tweet: 500 (since the characters mean of all tweets is about 250 characters and the std is around 150 characters, we used a maximum value of mean + 2 * std, as in the Normal Distribution).
- The *maximum number of tokens per tweet*: 100 (since the characters mean of all tweets is about 50 words and the std is around 28 words, we used a maximum value of mean + 2 * std, as in the Normal Distribution).
- In both cases, padding or truncation were applied accordingly + choosing from both the start and the end of each tweet (to cover a bigger range of “meaning”). Usual configuration was 50% from the start and 50% from the end of each tweet, but this feature is *customizable.*

Vocabulary:

- Character Vocabulary: using a minimum threshold to select the most frequent characters.
- Word Vocabulary: using a minimum threshold to select the most frequent words.
- All these thresholds are *customizable.*

3) NuSVR + Kernel Strings.

Kernel Strings**:** related paper https://www.aclweb.org/anthology/W18-3909.pdf (Andrei M. Butnaru and Radu Tudor Ionescu).
I used the p-grams relations from the range 3-8 with multiple strategies:

- Type of kernel strings: presence/spectrum/intersection
- Adding the matrices from the range 3-8 + normalization.
- Normalizing matrices from the range 3-8 + adding.
- Normalizing matrices from the range 3-8 + choose maximum value across each p-gram.

This kernels were then used as features for a NuSVR algorithm.

Final Comparison

Model type	**Mean Squared Error (MSE)**	**Mean absolute Error (MAE)**
WordCNN + Bert Embeddings	0.9287410769172326	0.7871208510219202
CharCNN + WordCNN + Kmeans (5 clusters)	0.8580104126574672	0.6258646145141811
NuSVR + String Kernel (Presence, 3-8 p-grams)	0.5929273601099221	0.5816183793905665
CharCNN + WordCNN regression	0.5758806003979809	0.5213326686183137
XGBoost Ensemble	0.49687972035091743	0.48919980437972493

These models are tested against the validation dataset provided.

The best model is the ensemble one using as features the predictions of several trained model (CharCNN + WordCNN, NuSVR with String Kernels). (but not with much when compared directly to the CharCNN + WordCNN regression one. This would imply a critical overhead that should be considered by the engineer designing models to run on real tasks, especially on embedded devices such as phones with limited resources available).