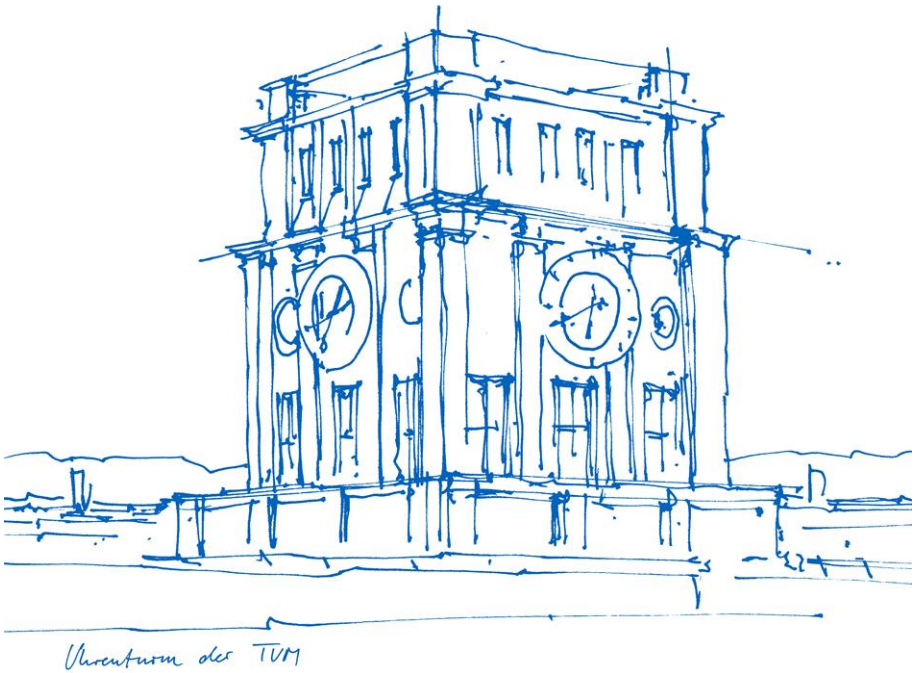
Exercises for
Social Gaming and
Social Computing
(IN2241 + IN0040) –
Exercise Sheet 5 -
Hate Speech
Detection



Exercise Content

[tabelle]

Exercise Sheet 5: Hate Speech Detection

- goal: analyse tweets by using a Pytorch Neural Network. Compare two models to see the impact of additional social information.
- data set: A collection of hand-labeled tweets by Waseem & Hovy [1]. Each tweet is either labeled 'sexism', 'none', or 'racism'. We will use its more than 16.000 tweets to train and test our models.

The Data: match information

- pickle_file.users_data: provides social information about the authors
  - *authorship.npy*: author for each tweet
  - *users_adjacency_matrix.npy*: the follower network of the authors
- tweets.csv: provides you with the labeled tweets from Waseem & Hovy

Task 5.1: Preprocessing

a) Encode the labels

In order for PyTorch to work with the labels, they need to have a specific format. Strings need to be replaced by numbers with an according mapping.

Map the labels from the `label_mapping = ['sexism' 'none' 'racism']` to a numeric vector.

b) Universal Sentence Encoder

Google's Universal Sentence Encoder (USE [4]) is a convenient way to map any type of sentence to a 512-dimensional vector. In these 512-dimensional vectors semantic meaning is encoded.

In this task you are suppossed to get a feeling for this type of embedding. Find a pair of sentences that are similar in their meaning but not syntactically. After that, think of two semantically very different sentences. Obtain the values for them and compare them.

Embed your two pair of sentences here.

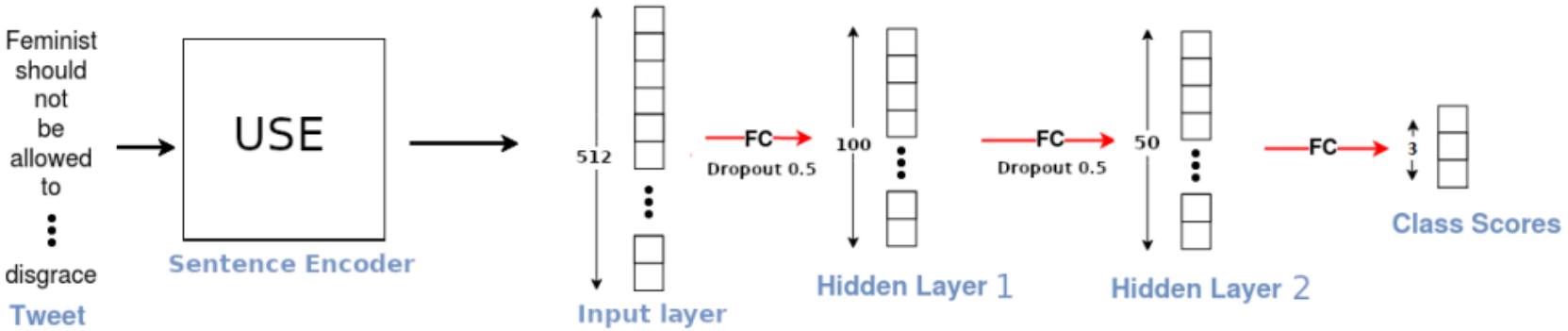
Task 5.2: Base Model

a) Base Model creation

In this code we create our base model and train it afterwards using the PyTorch library. We use a simple Neural Network to build this model. You can read about Neural Networks here in case you are not familiar with them. You can get a basic intuition for Neural Networks here [5].

For the base model we have our 512 dimensional input layer. Then we have a fully connected layer with 100 nodes and with a dropout rate of 0.5 is added. For now, you do not need to know what dropout is. After the dropout, another fully connected layer with 50 nodes is added and we once again add a 0.5 dropout rate. Our output layer has 3 nodes: One for "sexism", "none" and "racism". The computed values for these last 3 nodes correspond to the probability of belonging to either one of our categories.

Now compute the cross entropy loss and set the optimizer (with torch). For the learning rate use 0.00005.



b) Train-Test split for Base Model

Splitting our labeled data into a train test and validation set is a common practice.

- Train set: This set is used to train our model on. The model will try to learn from it.
- Validation set: This set is used to choose hyper parameters. Since creating good models requires to find the right parameters (e.g. what kind of activation function, how many epochs etc.) this set is used to maximize the performance of a model for a fixed choice of parameters.
- Test set: This set is used to evaluate our final model on. After the model has been trained and a final decision for hyper parameters has been made, the model will be evaluated on this set only. No more parameters should be changed after that.

This rather strange seeming approach helps to identify models that actually generalize well and not just perform very good because we adapted the parameters to maximize the performance on one particular set.

We will use 60% of our dataset to train our model (the train set) and the remaining 20% to evaluate our model (the test set).

Task

In order to feed the data into the model, we must create Dataset objects for it, allowing the creation of Dataloaders. The Dataset retrieves both the features and labels of the data. The dataset needs to have the following functions implemented: **init**, **len**, and **getitem**. The **getitem** function should return the tensorized encoding and label at the spezified position.

While training a model, we want to feed the data in batches and reshuffle it at every epoch to reduce model overfitting. Dataloaders offer an API to do that process.

```
# TODO: Create a CustomDataset class for our Tweet data
# Custom Dataset class
class CustomDataset(Dataset):
    def __init__(self, encodings, labels):
        #TODO

    def __len__(self):
        #TODO

    def __getitem__(self, idx):
        #TODO
```

Task

c) Train the Base Model

1. Specify the number of epochs as 20. Train and evaluate the model.

2. After you have looked at the graph, what do you think is an appropriate amount of `epochs` ? Briefly explain at which amount of epochs the model seems to be underfitting or overfitting and how this depends on the learning rate?

d) Evaluate the Base Model

The F1 score is a universal measurement of a test's accuracy. It is calculated as the harmonic mean of *precision* and *recall*.

- **precision** refers to the number of true positives divided by the number of all positives
- **recall** refers to the number of true positives divided by the number of relevant elements

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = \frac{tp}{tp + \frac{1}{2}(fp + fn)}$$

where

- tp = true positives
- fp = false positives
- fn = false negatives

1. Why would we prefer the F1 Score over only the precision?

2. Evaluate the text model with an F1 score.

Task 5.3: Preprocessing for Social Model

Now that we have evaluated our base model we can try to enhance it by using some sort of a social context. To do so, we are using our base model's prediction to compute an average hate score for each of the followers of an author. This means that for each author we take all his follower's tweets and predict the label. We then take the average of each prediction which results in our average hate score.

For each tweet we then not only feed in the tweet itself, but also the hate score.

a) Graph visualization & manipulation

Now we are going to plot the previously loaded adjacency matrix. Since we are going to feed the matrix to the Neural Network later, and because the user network is just a tiny subset of the whole Twitter network it is important to check if the network contains any useful information.

1. Plot the graph corresponding to the given adjacency matrix. **Note:** For better visualization, the nodes are color-coded based on their degree.

2. Describe what kind of communities you see in the graph and how they interact with one another.

3. Now let us try to actually calculate the number of communities within this graph. First, get rid of the uninteresting nodes that have zero or very few edges and just inspect the "core" graph. Expand on the code that you have written in the exercise above. **Hint**: You can do this by excluding all nodes with an `nx.eigenvector_centrality()` lower than 10^{-8}.

4. Do you think the social context could further improve our hate speech detection model? Find at least 2 pros and 2 cons.

Task

b) Employment of our Base Model to predict the hatefulness of an author's followers

Now that we have a trained model, we can use it to predict the hatefulness for any tweet. Therefore, we can use it to predict an average hate score for each follower of an author. This means that we predict the label for each tweet of an author's follower and then compute an average across all of these predictions.

1. Predict all encoded tweets with the Base Model

2. Now for every tweet of our dataset we need to compute its authors hate score. Therefore:

- First define a function `get_all_followers` that return all followers for a given user.
- Then create a list `their_followers` that contains all followers for each user.
- Now assign the hate predictions for each of all the followers tweets.
- Finally in `user_avg_score` compute the hate score for each user by averaging out all theirs followers' tweets' hate scores. If there are no followers' tweets, assign our pre-computed average values `default_hate_score`.

Task 5.4: Social Model

Now that we have our social context prepared, we can build and train our Social Model using that information.

a) Social Model creation

With our social context we have 2 separate networks:

- Our text network that processes the tweet (the same as the base model from before)
- Our hate score network that basically decides how important the average hate score for the classification is

Our 2 separate networks are concatenated and one last hidden layer with 100 nodes is added.

Task

b) Employment of our Base Model to predict the hatefulness of an author's followers

Now that we have a trained model, we can use it to predict the hatefulness for any tweet. Therefore, we can use it to predict an average hate score for each follower of an author. This means that we predict the label for each tweet of an author's follower and then compute an average across all of these predictions.

1. Predict all encoded tweets with the Base Model

2. Now for every tweet of our dataset we need to compute its authors hate score. Therefore:

- First define a function `get_all_followers` that return all followers for a given user.
- Then create a list `their_followers` that contains all followers for each user.
- Now assign the hate predictions for each of all the followers tweets.
- Finally in `user_avg_score` compute the hate score for each user by averaging out all theirs followers' tweets' hate scores. If there are no followers' tweets, assign our pre-computed average values `default_hate_score`.

Task 5.4: Social Model

Now that we have our social context prepared, we can build and train our Social Model using that information.

a) Social Model creation

With our social context we have 2 separate networks:

- Our text network that processes the tweet (the same as the base model from before)
- Our hate score network that basically decides how important the average hate score for the classification is

Our 2 separate networks are concatenated and one last hidden layer with 100 nodes is added.

Now define our new neural network `social_model` according to the graphic above. You can lookup most of the syntax in exercise ***5.2 a)***.

```
class SocialModel(nn.Module):
    def __init__(self, encoding_dim):
        super(SocialModel, self).__init__()
        #TODO

    def forward(self, text, followers):
        #TODO
```

b) Train-Test split for Social Model

Now split the data in Train/Val/Test 60/20/20 as seen in the Base Model. This time you have to create an additional set for the Hate Score.

Once again we need to create adequate Datasets and Dataloaders. **getitem** should return the tensorized encodings, hatescore and labels.

```
# Custom Dataset class
class SocialDataset(Dataset):
    def __init__(self, encodings, hatescore, labels):
        #TODO

    def __len__(self):
        #TODO

    def __getitem__(self, idx):
        #TODO
```

c) Train and Evaluate the Enhanced Model

Once again, train and evaluate the model with a F1 Score. Use 20 `epochs`.

Task 5.5: Discussion and comparison

- Compare the performances of our two models in your own words
- Why do you think it improved?
- Can you think of any other social context to further improve our model?

Submitting your solution

- work by expanding the .ipynb iPython notebook for the exercise that you downloaded from Moodle
- save your expanded .ipynb iPython notebook in your working directory
- submit your .ipynb iPython notebook via Moodle (nothing else)
- remember: working in groups is not permitted. Each student must submit their own .ipynb notebook!
- we check for plagiarism. Each detected case will be graded with 5.0 for the whole exercise
- deadline: check Moodle

Citations

[1] Waseem, Z., & Hovy, D. (2016). Hateful symbols or hateful people? Predictive features for hate speech detection on Twitter. In Proceedings of the naacl student research workshop (pp. 88-93).