23.12.2019

Data Mining Homework 2 Report

Felicia Müller – X1080023

In this report all important steps for the Kaggle-Competition submissions are mentioned and explained. First, the preprocessing steps are described and afterwards the generation of the models is presented.

Preprocessing

- Hashtags were put at the beginning of each sentence
    - First models were trained without considering the hashtags
    - Good improvement trough considering them
- Convert to lower case
- Remove stop words
- Remove numbers
- Stemming/lemming → both were tried, stemming lead to better results
- BOW, TFIDF with different number of features (500, 1000 and 1500) → better results with BOW and 1500 features

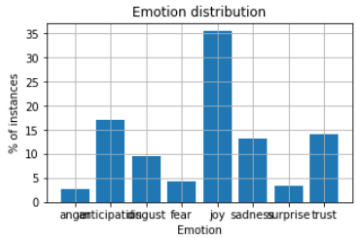

Figure 1: Histogram of training data.

The code of this preprocessing steps, which were done with a Kaggle notebook, can be found in the attachements:

KaggelCompetition_Preprocessing.ipynb

First Models

- Neural network from class
    - Different layers (duplication of existing layers) → little influence on the result
- Another neural network (Model 2)
    - Model:

      ```
      model.add(Dense(500, input_shape=(input_shape)))
      model.add(Dense(64))
      model.add(Dense(64), activation='relu')
      model.add(Dense(64))
      model.add(Dense(64, activation='relu'))
      model.add(Dense(8))
      model.add(Dense(8, activation='softmax')
      ```

    - Adjust for imbalanced dataset (see figure 1):
        - Random Under/Over Sampling was applied to adjust for this → worse results
        - Class weights in neural networks were applied → improved results
        - Recall and F1-Score were applied as metrics additional to accuracy → better results
    - Model parameters
        - Sgdm vs. adam → adam performed slightly better
        - Different dropout rates (0.3 0.5) → dropout led to worse results
        - Different layers (adding another dense layer with 256 units, adding a layer with 512 units, adding one with 512 units, removing the one with 64 units → all those adjustments lead to (slightly) worse results)
        - Elu and relu as activation functions of the different layers → relu slightly better than elu

- Bert Model with 12 layers according to https://appliedmachinelearning.blog/2019/03/04/state-of-the-art-text-classification-using-bert-model-predict-the-happiness-hackerearth-challenge/
    - Code for the classifier was adjusted to distinguish between eight different groups
    - Training took too long → I broke up the training and used a created checkpoint for the classification of the test-set
    - Unfortunately, the classification took too long, thus I skipped it
    - Maybe this model would have led to good results but with my laptop it was not manageable (the complete training and classification would take at least several days and as I did not consider it as useful to only focus on this approach I decided to use my computer resources for the models mentioned above)

Final Model: Recurrent neural network with LSTM

- Model:

```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(8, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer="adam",metrics=['accuracy'])
print(model.summary())
```

- Metric “accuracy” lead to better results than recall with f1-score
- More epochs lead to a worse result
- Increasing the dropout rate lead to worse results
- Sentences as input with keras preprocessing tokenizer lead to better results than the preprocessing described at the beginning and used in the former models
- Different batch sizes: 32/64 almost indifferent ⟶ 64 was kept because it needs much less training time
- Different number of relevant words: 40000 instead of 50000 ⟶ slightly worse, so the considered words are important
- Model complexity reduction: 80 instead of 100 in LSTM layer because after the first epoch the training accuracy is much higher than the testing accuracy and the training loss is much less than the testing loss. Thus, a less complex model might represent the data better but the result got slightly worse
- Regularization to reduce difference between testing and training but results got worse (L2 regularization)

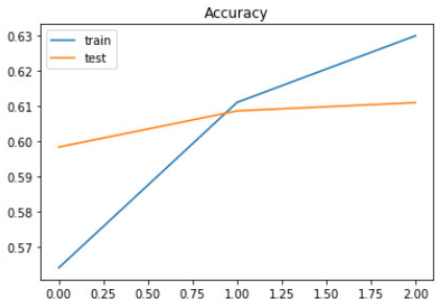

Figure 2: History of model accuracy.

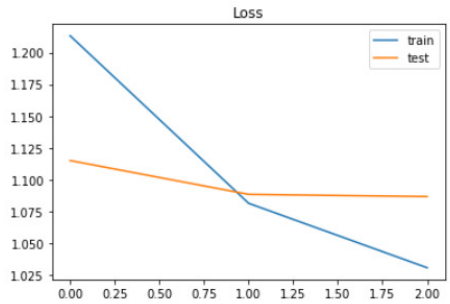

Figure 3: History of model loss.

The python code of the best model is attached in the file LSTM_Model.ipynb and was executed with jupyter notebook.