Project Report: Irony Detection with Domain Adaptation

Anne Beyer (11802814)
Deep leaning for Natural Language Processing (WS17/18)
CIS, LMU

1 Task

This project examined different approaches for combining two datasets to improve the performance in an irony detection task.

In *irony detection*, a model is trained to predict whether a given utterance is ironic (i.e. meaning the opposite of what was actually said) or not[1]. The task of *domain adaptation* describes a scenario where data from a certain distribution is used for training a model whose performance is then evaluated on data from a different distribution. In this project, the performance in classifying reddit comments was examined for models using different amounts of additional labeled Twitter data.

2 Data

The two datasets used in this project are:

- **SemEval:** This dataset is taken from the SemEval 2018 Task 3a[2]. It consists of 1911 ironic and 1923 non-ironic tweets.
- **Kaggle:** This dataset contains reddit comments[3] and their human annotations, yielding 537 ironic and 1412 non-ironic comments[4].

The SemEval data was used as out-of-domain data and the performance of the different models was evaluated on a held-out test set from the the Kaggle data. Five different data setups were examined:

1. **Out-of-domain:** This model is trained solely on the SemEval data.
2. **Joint:** This model is trained on a concatenation of both training sets.

[1]There are more elaborate tasks which also define different granularities of irony, but this project was just concerned with a binary classification

[2]`https://competitions.codalab.org/competitions/17468`

[3]`https://www.kaggle.com/rtatman/ironic-corpus`

[4]Without the development and test sets, there were around 1,000 samples in the kaggle training corpus.

3. **Weighted:** This model uses the same concatenation of the data, but with additional weights to adjust the imbalance in the amount of data[5].

4. **Continued:** This model is initialized with the weights from the out-of-domain model and then trained further only on the Kaggle data.

5. **In-domain:** This model is trained from scratch solely on the Kaggle data.

As the Kaggle dataset is unbalanced, the data for models 2, 3, 4 and 5 was balanced using the *compute_class_weight* function from scikit-learn[6].

3 Model Architecture

The architecture uses the keras Sequential model API and implements a simple bi-directional recurrent neural network based on LSTMs. The number of epochs for each model is determined by early stopping based on the validation loss on the development set. Except for this, all models were trained using the same set up[7].

4 Results

Figure 1 presents the results of the different models sorted by the proportion of out-of-domain data in the training set along the x-axis[8]. The performance is measured in terms of accuracy and F_1 score with precision and recall plotted as well for clarity.

Before these numbers can be interpreted, it is important to note the following: As the ratio between ironic and non-ironic examples is unbalanced in the kaggle corpus, this also holds true for the test set. A closer examination showed that the test set contains about 70% non-ironic segments. This means that a model which only predicts the label *non-ironic* will by default reach an accuracy score of 70%. Therefore, the evaluation also includes the F_1 score.

The out-of-domain model performs best in terms of the F_1 score, but worst in terms of accuracy. The combination of highest recall and lowest precision values shows that this model predicts most of the ironic comments correctly, but at the same time generates many false positives. Adding in-domain data leads to a substantial improvement in terms of accuracy (but see the note above) but also decreases the recall values drastically. This is especially the case for the joint and the in-domain model, with the latter also scoring worst in terms of recall and thus F_1 scores. The best performing models are the weighted and the continue model, even though their F_1 scores are still fairly moderate.

[5]Using the keras option *sample_weight* to balance out the different sizes of the two corpora yielded surprisingly bad results. To double check this, I trained another model by manually adding the in-domain data four times to the training data. This led to a huge improvement in the results and suggests that there was a problem in the implementation. The results reported in Section 4 are the ones obtained by this improved model

[6]`http://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html`

[7]The implementation details can be found in the source code.

[8]The continued model is put in for comparison after the weighted model, assuming that with early stopping the model uses as much of the in-domain data "as it needs", even though the approach to combining the data is a different one in this case.

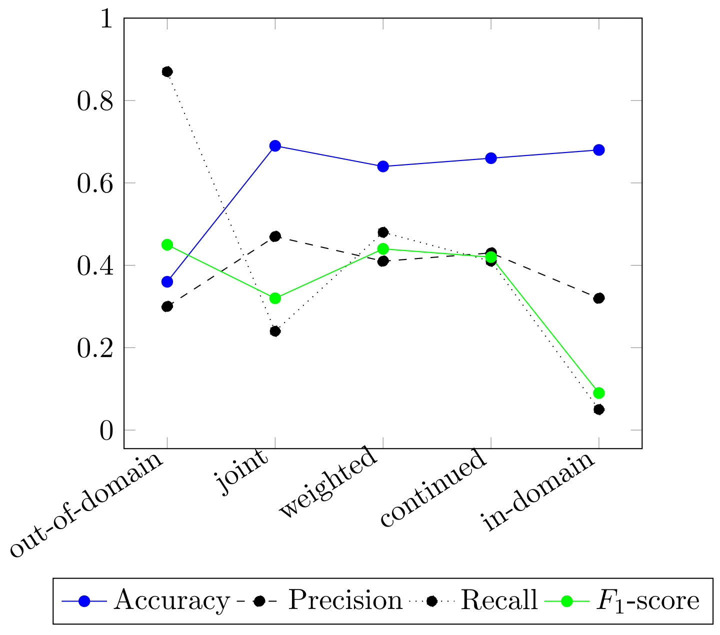

Figure 1: Evaluation of models using different amounts of out-of-domain data[9]

5 Discussion

The results of the evaluation show that combining the datasets definitely helped.

The amount of in-domain data was too small to train a useful model on it alone. The out-of-domain model was trained on four times as much data, but due to the differences between tweets and reddit comments, this model produced too many misclassifications. The best results were achieved by the two models that compensated for the differences in the training data by either counterbalancing the imbalance in the size of the training sets (i.e., the weighted model) or training only on the in-domain data but initializing the model with the weights from the out-of-domain model (i.e. the continued model).

In general, the results still need further improvement. One possible approach would be to test different model architectures, e.g. using a CNN instead of an RNN. As the training data was very small, another possibility would be to additionally use pre-trained word embeddings.

Here, the focus was on the combination of different datasets. A related project showed that using regularization techniques can also help to further improve the results of the best performing model from this project.

[9]Note that the results reported here differ from the ones presented earlier. Using early stopping with *patience* = 1 lead to very few training epochs (2–5). To get a better understanding of what these models learn the scores reported here result from an increase of *patience* to 3 (and the adapted data for the weighted model).