Natural Language Processing

Aspect-Term Polarity Classification in Sentiment Analysis

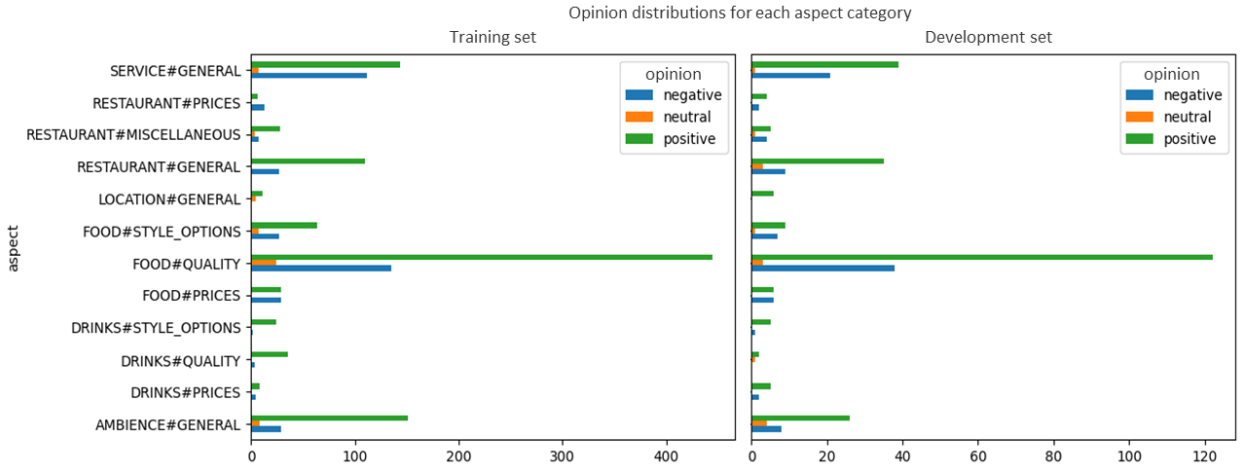
by: Ian Moon, Adel Remadi, Lasse Schmidt

within: MS Data Sciences & Business Analytics

at: CentraleSupélec & ESSEC Business School

1. Introduction

The aim of this project has been to implement a classifier that predicts opinion polarities (positive, negative or neutral) for a given set of aspect categories related to a sample of restaurant reviews. The dataset of interest was composed of a training set and a development set, where there is high imbalance both in terms of the number of reviews per polarity as well as the number of reviews per category.



2. Implementation Details

The input data contains four elements: the aspect category, the target term, its position and the review itself. These elements were preprocessed by concatenating them into a single string. This string was enriched with model-specific separators and marker tags in order to provide some basic structure as well as highlight the target term's position in the sentence. Below is a concrete example from the training dataset (using respective marker tokens for RoBERTa):

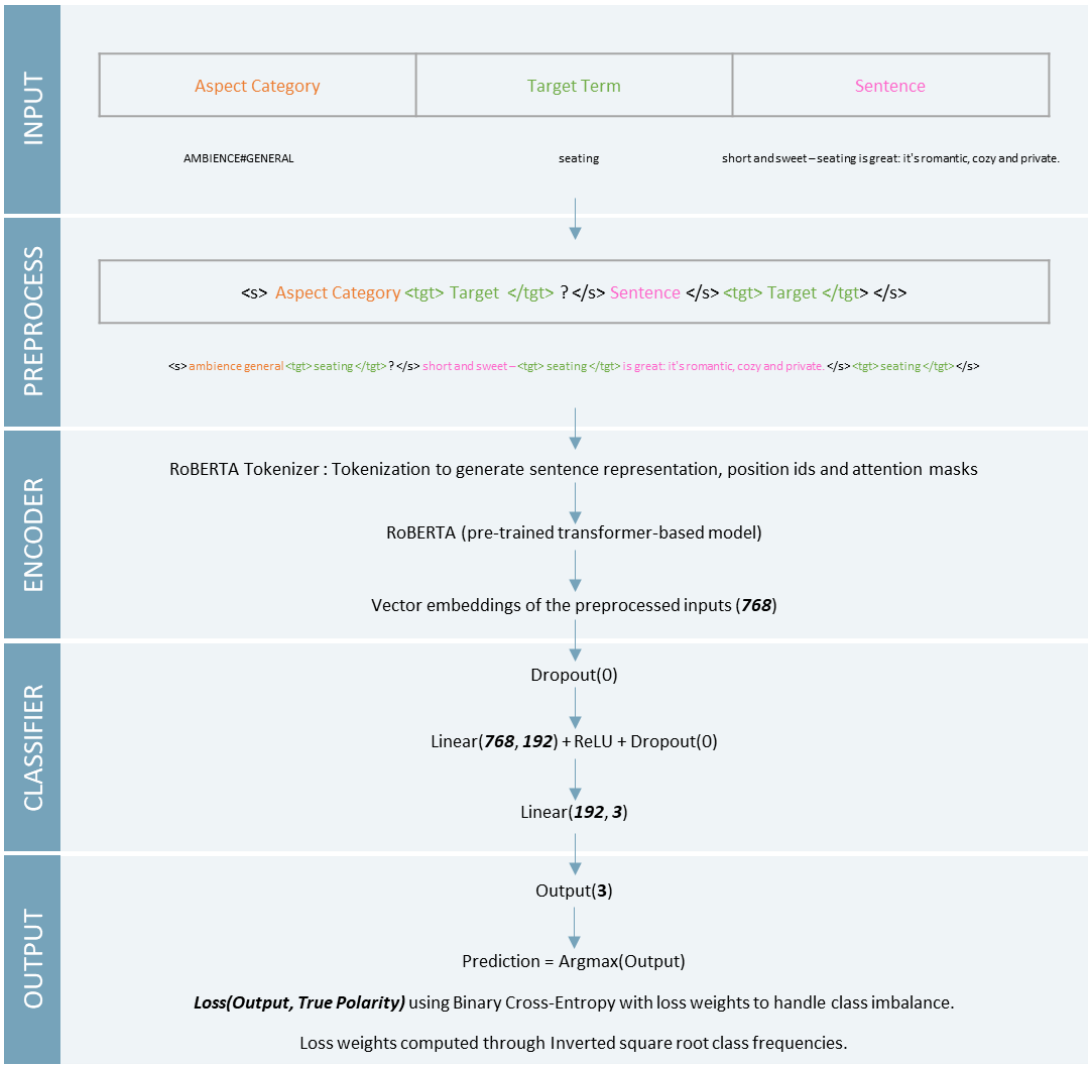
<s> ambience general <tgt> seating </tgt> ? </s> short and sweet – <tgt> seating </tgt> is great: it's romantic, cozy and private. </s> <tgt> seating </tgt> </s>

Our submitted model is based on the combination of a pre-trained encoder-only transformer, namely RoBERTa, to generate the vector embeddings of the preprocessed strings and a classifier network to predict the polarity based on the vector embedding. RoBERTa is an optimized version of BERT (Bidirectional Encoder Representations from Transformers) and follows a similar input structure (with small differences in tokenization and special tokens).

The final architecture of the classifier network as well as all the hyperparameters were determined in a series of grid-searches using the *raytune* library. Among the tested configurations, the optimal architecture for the classifier were 2 linear layers with 192 hidden units, ReLU as the activation function, and no Dropout.

Another critical aspect of the pipeline was the choice of the loss function. We evaluated the performance using Binary Cross Entropy Loss, Focal Loss, and Top-K Loss. Eventually, the Binary Cross Entropy was selected. In order to tackle the class imbalance mentioned in the introduction, loss weights were computed, and the optimal results were obtained with inverted square root class frequency.

The following diagram illustrates the overall pipeline that was implemented:

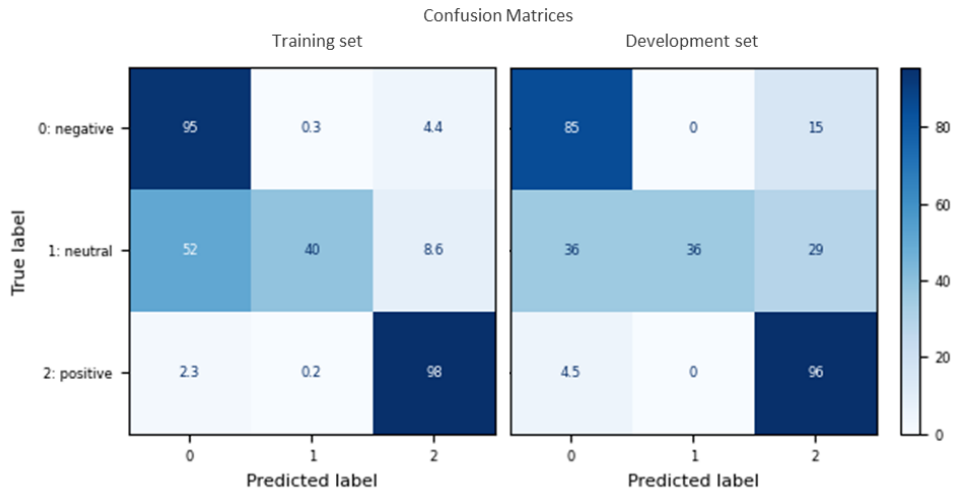


3. Main Results

The selected model described above has obtained the following performance on the Dev Set (5 runs):

Pre-trained Model	Machine	Number of epochs	Average Dev Accuracy (5 runs)	Standard Deviation	Total Runtime (5 runs)
Roberta-base	Local Machine (RTX 3070 8GB)	20	90.32	0.57	**20 min** (1176 s)

The following confusion matrices show that the most misclassifications occur for the neutral polarity. This makes sense as it will likely be the most ambiguous class (no extreme expressions of polarity) and as we have by far the fewest number of examples for this polarity: only 58 of over 1,500 reviews in the training data relate to neutral polarity. In addition, the proportion of negative comments misclassified as positives is higher on the development set. Other than these two aspects, the performance of the predictions on the train and dev set show similar patterns, which seemed to indicate a good potential for generalization.



Several variants of the model presented in section 2 were tested and achieved very good results, as illustrated in the table below. They even outperformed our submission, but were eventually discarded to ensure a better trade-off between performance and runtime. As an example, using RoBERTa-large as a pre-trained model, we obtained the highest performance of **92.39** with low standard deviation. However, the runtime was much longer, reaching about 2.5 hours to complete the 5 runs on Google Colaboratory.

Pre-trained Model	Machine	Number of epochs	Average Dev Accuracy (5 runs)	Standard Deviation	Total Runtime (5 runs)
Roberta-base	Local Machine (RTX 3070 8GB)	10	88.56	0.97	11 min (650 s)
Roberta-large	Google Colaboratory (Tesla T4 14GB)	10	92.02	0.81	~ 1h15 (4451.22 s)
Roberta-large	Google Colaboratory (Tesla T4 14GB)	20	**92.39**	**0.27**	~ 2h30 (9260.80 s)

4. Hyperparameter tuning

As mentioned previously, the obtained results were achieved through a series of grid-searches with the *raytune* library that covered various hyperparameters. Below you can see the 'config' dictionary which we used to pass the hyperparameters of each run to the classifier:

```
36          self.config = {
37              # basic infos
38              "verbose": False,
39              "max_epochs": 20,
40              "batch_size": 32,
41
42              # data preprocessing
43              "input_enrichment": "short_question_sentence_target",
44
45              # pre-trained language model (transformer)
46              "plm_name": "roberta-base",
47              "plm_freeze": False,
48
49              # classifier (linear layers)
50              "cls_depth":          2,
51              "cls_width":          192,
52              "cls_activation":     "ReLU",
53              "cls_dropout_st":     0, # maybe try out 0.1 sometime
54              "cls_dropout_hidden": 0,
55
56              # optimizer
57              "lr": 5e-6,
58              "wd": 15e-3,
59
60              # scheduler
61              "lr_s": "linear",
62              "warmup": 0,
63
64              # loss function
65              "crit": "BCE",
66              "crit_w": "invSqrtClassFreq",
67          }
```

To illustrate the amount of work we put into this step, let us explain some hyperparameters and our choices.

input_enrichment: refers to the preprocessing step described in section 2. Several enrichments were considered, and their performances were tracked and compared to identify the most appropriate one. For example, other options involved the use of natural language questions to encode the aspect category (such as 'What is the ambience like regarding the *$target_term$* ?').

plm_name: The pre-trained models that were considered for tuning are 'bert-based-cased', 'roberta-base' and 'roberta-large'. The whole pipeline, from the preprocessing step to the prediction was designed so that it could handle any choice of model among those three.

Architecture of the classifier (linear layers on top of Language Model): As mentioned in section 2, the classifier's architecture was determined through several grid-searches on different hyperparameters such as depth, width, which activation function to use, and whether or not to apply dropout. Additionally, the choice of the optimizer, its parameters (such as learning rate and weight decay) and its scheduling was also considered.

Loss function: Three loss functions were evaluated, namely Binary Cross-Entropy (BCE), Focal Loss and Top-k loss. BCE showed the most promising results and was then selected in the final model. The class imbalance of the dataset was addressed through the computation of loss weights aiming at steering the model to classify correctly the underrepresented classes. Three options were considered in the tuning

phase: unweighted (standard), inverted class frequency and inverted square root class frequency. Unweighted BCE loss led to the neutral class not being predicted at all, which is clearly not optimal. Then we tried out the inverse class frequency, which led to weights of roughly (3, 24, 0.5) for (neg, neutral, pos). This led overall to better results but misclassifications on the neutral class were so heavily penalized that model performance still suffered. Lastly, we tried out square root inverse class frequency, which led to weights of roughly (1.69, 5, 0.65) for (neg, neutral, pos). Here, we still penalize misclassification errors on the neutral class the most, however the coefficients are much smoother overall. This generated consistently better results.