Sentiment analysis with Yelp and Bert

Sentiment Analysis is one of the most common NLP (Natural Language) applications. With machine learning you can train models based on textual datasets that can identify or predict the sentiment in a piece of text, like e.g. "negative" or "positive". In this blog we are going to describe how you can **train** such a model with practical example and then create a **REST interface** which you can use to predict the sentiment of a text. The source code for this exercise can be found on Github.

5 Star Sentiment Analysis

The typical Sentiment Analysis models and examples in the "blogosphere" use two categories, so we looked for a variation of this "negative" / "positive" models and tried to find a dataset which uses 5 star rating with the goal of training an ML model that is able to categorise the sentiment based on five stars - thus giving a more nuanced idea about the sentiment. We have found a publicly available dataset by Yelp, a review website.

Yelp Data

Yelp provides a JSON based review dataset which multiple fields out of which we have 2 that we used for training our sentiment analysis model:

text	Normal English text, like e.g: "As someone who has worked with many museums, I was eager to visit this gallery on my most recent trip to Las Vegas. When I saw they would be showing infamous eggs of the House of Faberge from the Virginia Museum of Fine Arts (VMFA), I knew I had to go! ..."
stars	Numbers 1 to 5 corresponding to the stars

The full documentation of the dataset we used is on this page:

review.json

Contains full review text data including the user_id that wrote the review and the business_id the review is written for.


```
{
    // string, 22 character unique review id
    "review_id": "zdSx_SD6obEhz9VrW9uAWA",

    // string, 22 character unique user id, maps to the user in user.json
    "user_id": "Ha3iJu77CxlrFm-vQRs_8g",

    // string, 22 character business id, maps to business in business.json
    "business_id": "tnhfDv5Il8EaGSXZGiuQGg",

    // integer, star rating
    "stars": 4,

    // string, date formatted YYYY-MM-DD
    "date": "2016-03-09",

    // string, the review itself
    "text": "Great place to hang out after work: the prices are decent, and

    // integer, number of useful votes received
    "useful": 0,

    // integer, number of funny votes received
    "funny": 0,

    // integer, number of cool votes received
    "cool": 0
}
```


As you can see we used only a portion of the dataset. It would also be possible to create a model that predicts funny, cool or useful content. We will have a look at this in an upcoming blog.

Model and Libraries

Sentiment Analysis models are these days typically deep learning models. ML practitioners also do use existing pre-trained models as the starting point for their training. This technique is referred to as transfer learning.

For our 5 star sentiment analysis exercise we have chosen the BERT model. BERT is a neural network architecture which was created and published in 2018 by Google researchers and delivers state-of-the-art performance in many NLP tasks.

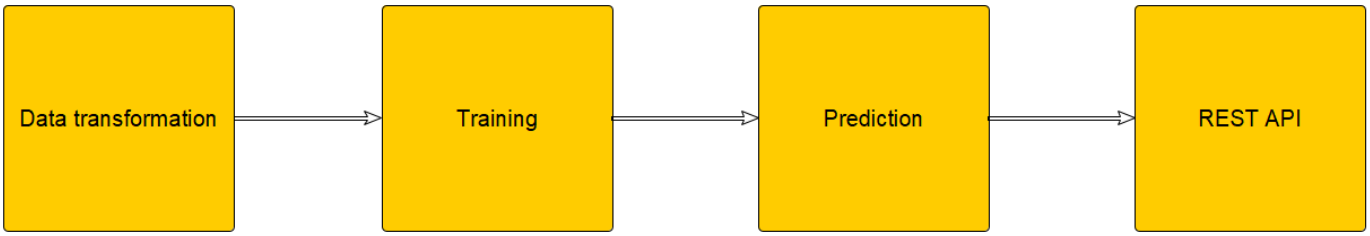
Our language of choice for ML is Python that has another three of your favourite libraries used in this exercise:

Pytorch	awesome deep learning library by Facebook that allows to train and also to perform inference based on neural networks
Hugging Face Transformers	library that provides a very large amount of pre-trained neural networks and also tools for training and using NLP models.
Flask	A Python micro web framework, great for creating quickly REST interfaces

The Hugging Face Transformers provides a pre-trained version of BERT model based on cased text which we used in our training.

Steps for creating the ML Model and REST

This exercise has 4 steps:

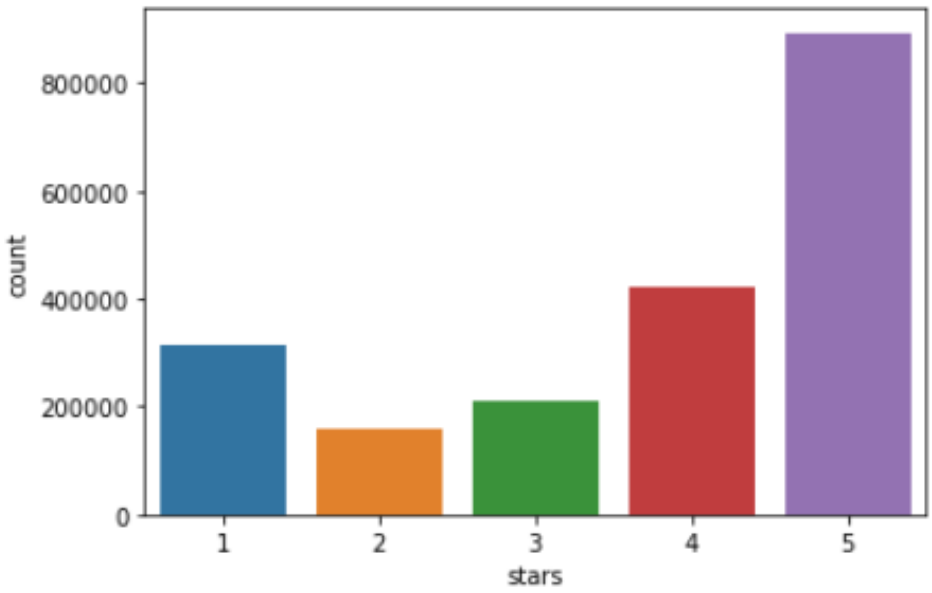


Data Transformation

The original Yelp reviews come in JSON format and also with some unnecessary fields. We converted the original format to a Pandas dataframe with only two fields, text and stars:

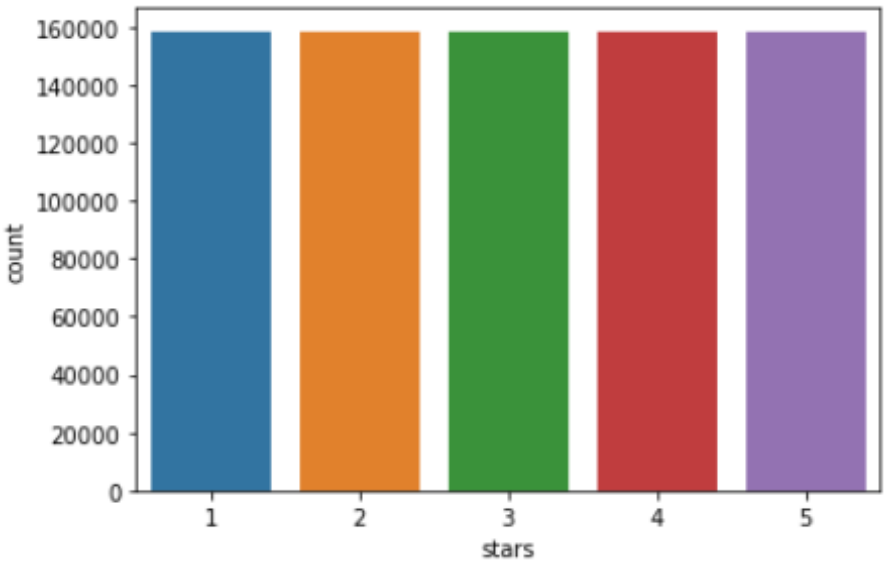
	text	stars
0	I am actually horrified this place is still in...	1
1	I love Deagan's. I do. I really do. The atmosp...	5
2	Dismal, lukewarm, defrosted-tasting "TexMex" g...	1
3	Oh happy day, finally have a Canes near my cas...	4
4	This is definitely my favorite fast food sub s...	5
...	...	...
1999994	This place is excellent. I never knew about th...	5
1999995	An all-time favorite for me and my wife. We g...	5
1999996	I have been coming here for over a year now fo...	5
1999997	Worst customer service on the planet! Came in ...	1
1999998	Thanks Dr. Nelson for helping me too feel bett...	5

The dataset is also unbalanced, i.e: some categories are over- and others under-represented. Here is a graph of the distribution of a sample of 2 million records:



As you can see people like to give more positive than negative reviews; one star reviews are more common than 2 and 3 star reviews.

An unbalanced dataset might lead to a biased classifier which during inference time gives preference to the class with the highest frequency, so we decided to create a dataset with the same number of records per class. And after removing records from different classes we got this distribution:

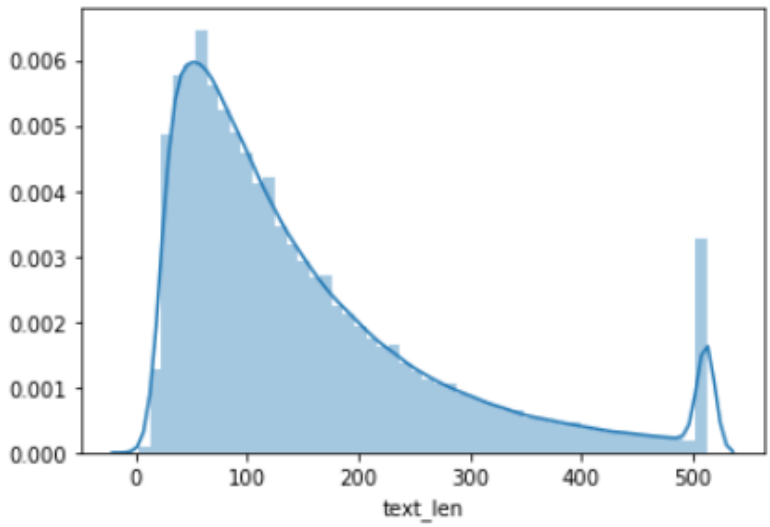


The final training set contained 792786 records out of a sample of 2 million records out of the original 8 million records. We serialize the Pandas dataframe with all records in a Python pickle file.

The Python Jupyter notebook with the initial data transformation is available on Github.

Training

In the training notebook (also available on GitHub) we read the previously serialized (or pickled) pandas dataframe and we checked first the distribution of the lengths of the Yelp reviews. It seems that most reviews have around 80 words or so. But there are a good amount of review that have more that 512 words. Overall the distribution of the lengths of words in review should be skewed to the right:



Based on this distribution we decided the set the maximum length of the BERT tokenizer to 512. As it seems this number is also the default for most tokenizers provided by the Transformers library:

```
[14]: tokenizer.max_model_input_sizes

[14]: {'bert-base-uncased': 512,
       'bert-large-uncased': 512,
       'bert-base-cased': 512,
       'bert-large-cased': 512,
       'bert-base-multilingual-uncased': 512,
       'bert-base-multilingual-cased': 512,
       'bert-base-chinese': 512,
       'bert-base-german-cased': 512,
       'bert-large-uncased-whole-word-masking': 512,
       'bert-large-cased-whole-word-masking': 512,
       'bert-large-uncased-whole-word-masking-finetuned-squad': 512,
       'bert-large-cased-whole-word-masking-finetuned-squad': 512,
       'bert-base-cased-finetuned-mrpc': 512,
       'bert-base-german-dbmdz-cased': 512,
       'bert-base-german-dbmdz-uncased': 512,
       'TurkuNLP/bert-base-finnish-cased-v1': 512,
       'TurkuNLP/bert-base-finnish-uncased-v1': 512,
       'wietsedv/bert-base-dutch-cased': 512}
```

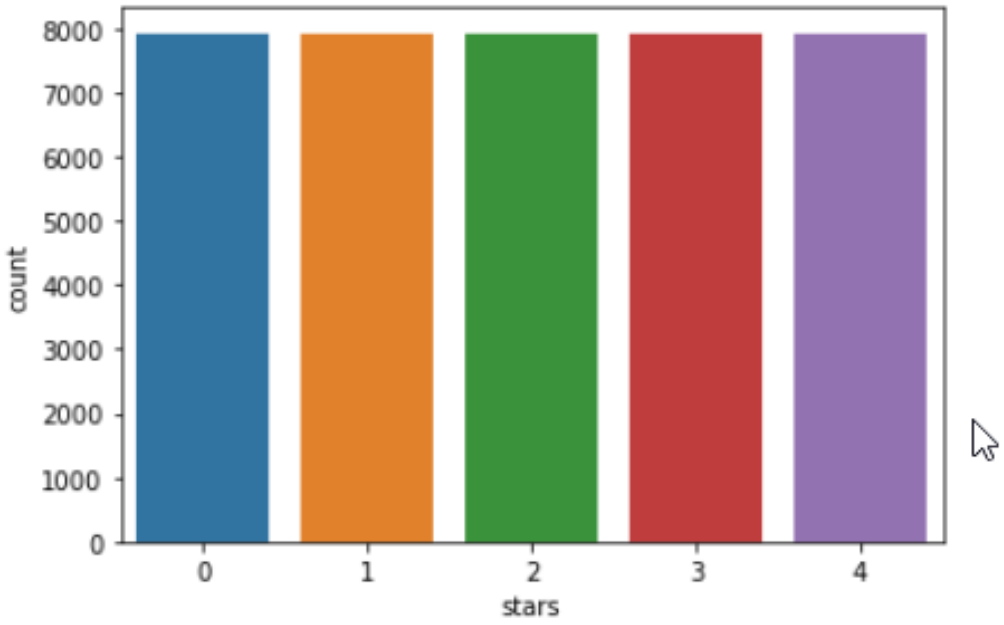
After our mini text length distribution analysis we proceeded to create the Yelp data set class that uses the Transformers library's tokenizer:

```
In [13]: class YelpDataset(Dataset):
             def __init__(self, reviews, targets, tokenizer, max_len):
                 self.reviews, self.targets,  self.tokenizer, self.max_len = reviews.to_numpy(), targets.to_numpy(), tokenizer, max_len

             def __len__(self):
                 return len(self.reviews)

             def __getitem__(self, item):
                 review = self.reviews[item]
                 tokens = self.tokenizer.encode_plus(
                     review,
                     add_special_tokens=True,
                     max_length=self.max_len,
                     return_token_type_ids=False,
                     pad_to_max_length=True,
                     return_attention_mask=True,
                     return_tensors='pt'
                 )

                 return {
                     'review': review,
                     'input_ids': tokens['input_ids'].flatten(),
                     'attention_mask': tokens['attention_mask'].flatten(),
                     'target': torch.tensor(self.targets[item]).long()
                 }
```

We have also split the data in a train, validation and test sets using stratification to avoid unbalanced sets.

```
In [15]: train_df, test_df = train_test_split(balanced_simplified_reviews, test_size=0.1, random_state=RANDOM_SEED,
                                              stratify=balanced_simplified_reviews.stars.values)
         valid_df, test_df = train_test_split(test_df, test_size=0.5, random_state=RANDOM_SEED, stratify=test_df.stars.values)
         train_df.shape, test_df.shape, valid_df.shape

Out[15]: ((713507, 3), (39640, 3), (39639, 3))
```

So again we kept the datasets balanced:



Next we created the dataloaders for our training, validation and test sets with a batch size of 16:

```
In [18]: BATCH_SIZE=16

         train_dl = create_data_loader(train_df, tokenizer, config.MAX_LENGTH, BATCH_SIZE)
         test_dl = create_data_loader(test_df, tokenizer, config.MAX_LENGTH, BATCH_SIZE)
         valid_dl = create_data_loader(valid_df, tokenizer, config.MAX_LENGTH, BATCH_SIZE)
```

As our model we have used a model provided by the Transformers library: *BertForSequenceClassification*. This is how we instantiate the model using a factory method:

```
In [22]: model = BertForSequenceClassification.from_pretrained(config.PRE_TRAINED_MODEL_NAME, num_labels = NUM_CLASSES,
                                                                output_attentions = False, output_hidden_states = False)
         model = model.to(device)
```

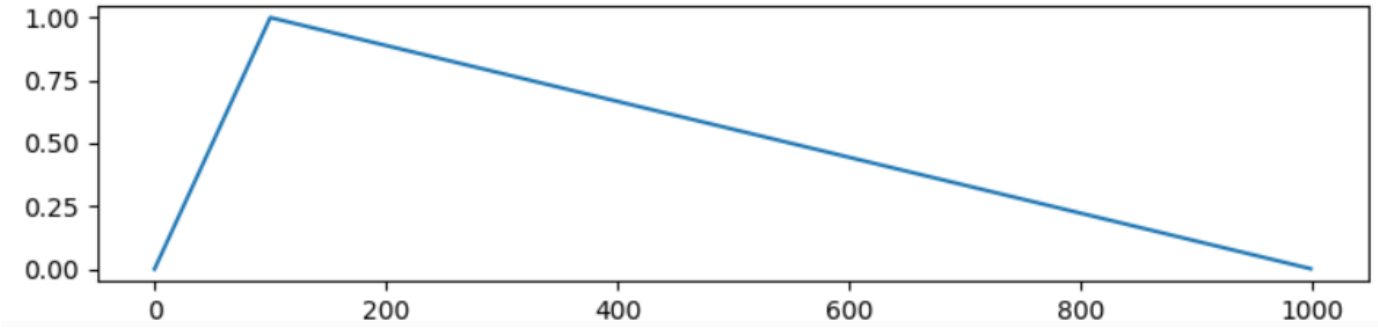
The optimizer AdamW of the Transformers library with a learning rate of 2e-5 (0,00002) was used:

```
In [25]: # AdamW Adam algorithm with weight decay fix
         optimizer = AdamW(optimizer_parameters,
                           lr = 2e-5, # args.learning_rate - default is 5e-5, our notebook had 2e-5
                           eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                          )
```

Also a learning rate scheduler (*transformers.get_linear_schedule_with_warmup* provided by the Transformers library) was used:

```
In [26]: EPOCHS = 3

         total_steps = len(train_dl) * EPOCHS

         # Create the learning rate scheduler.
         scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps = 0, num_training_steps = total_steps)
```

This scheduler changes the learning rate in this way:



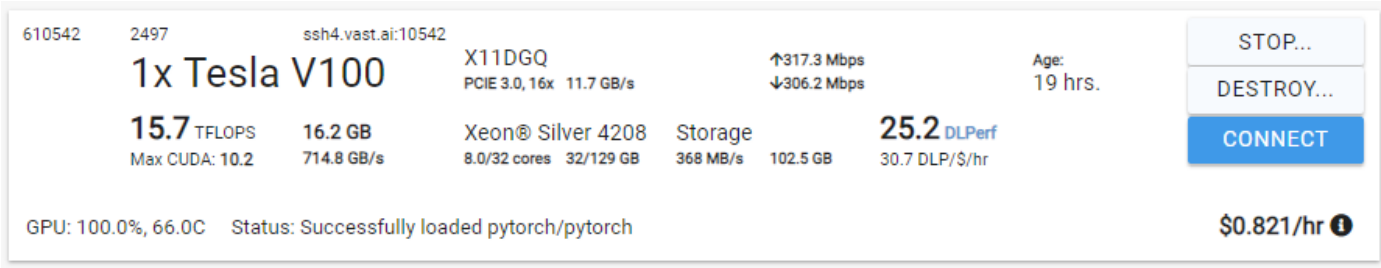
And finally we created a very standard training loop and evaluation loops. See the source code on Github for more information.

As metrics we just used the accuracy metric:

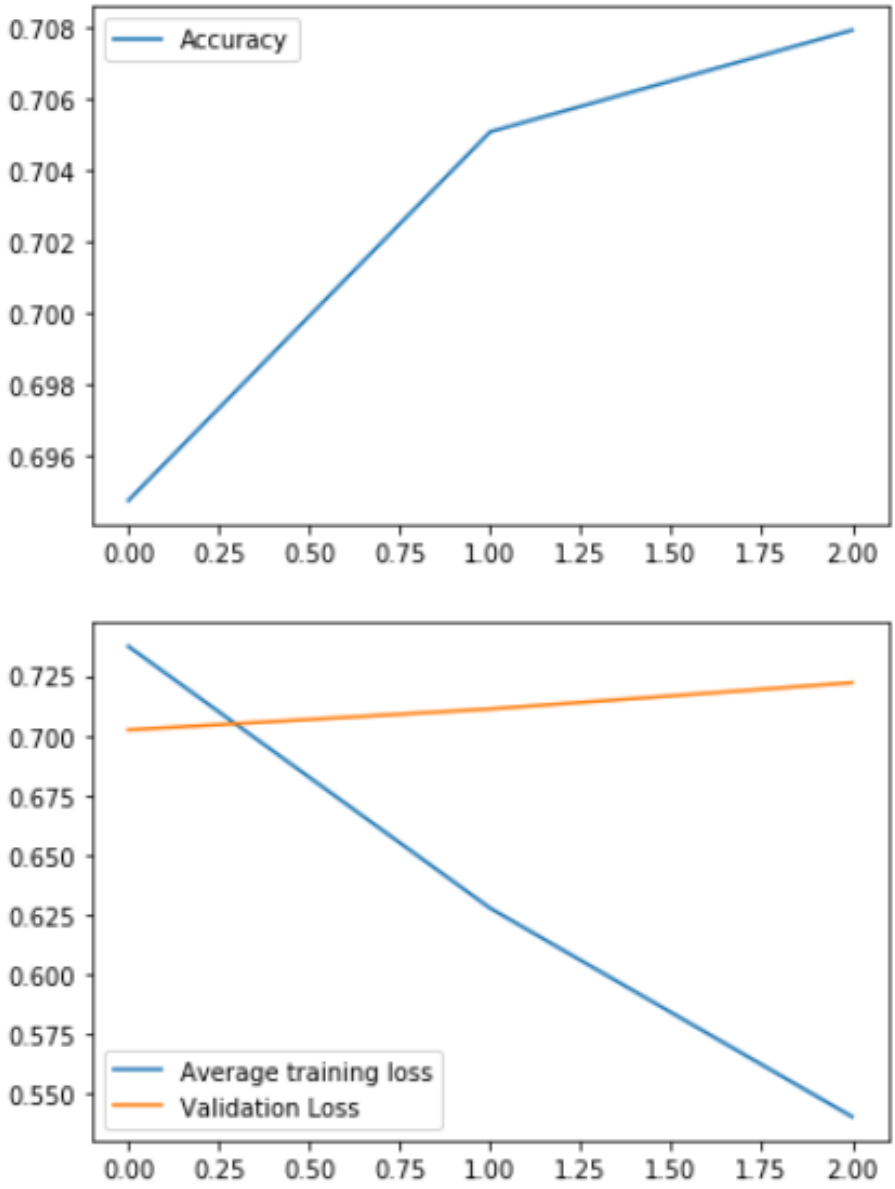
```
In [27]: # Function to calculate the accuracy of our predictions vs labels
         def flat_accuracy(preds, labels):
             pred_flat = np.argmax(preds, axis=1).flatten()
             labels_flat = labels.flatten()
             return np.sum(pred_flat == labels_flat) / len(labels_flat)
```

Training results

We have trained on a GPU machine provided by vast.ai with a Tesla V100 with 16GB of RAM:



The best accuracy we got was 0.71 which means that the star prediction on the test set was correct in 71% of cases. This accuracy was achieved on the second epoch of the training that took almost 20 hours.



Prediction / Inference

After the training we have simply "pickled" the best trained model and created a simple Jupyter notebook that we used to check our predictions. This notebook is available on Github.

In this notebook we created a predict method which encodes the input sequence using the same tokenizer we used during training and then uses the encoding as input of the trained model. The trained model outputs a tensor with five elements that represents the probabilities for each class.

This is the predict method:

```
[18]: def predict(sequence='I love you a lot. You are really great. You are wonderful and awesome.'):
          encoded = encode(sequence)
          with torch.no_grad():
              output = model(encoded['input_ids'].cpu(), token_type_ids=None, attention_mask=encoded['attention_mask'].cpu())[0]
              pred_flat = np.argmax(output, axis=1).flatten()
              sig_factor = torch.sigmoid(output) / torch.sigmoid(output).sum()
              return {'proportional':  sig_factor.numpy().tolist(), 'sigmoid': torch.sigmoid(output).numpy().tolist(),
                      'stars': pred_flat.item() + 1, 'raw': output.numpy().tolist()}
```

Here are some examples of how you can use this model with simple sentences in a notebook:

```
[10]: predict('This is really terrible. Just avoid it')

[10]: {'proportional': tensor([[0.4823, 0.4148, 0.0824, 0.0080, 0.0125]]),
       'sigmoid': tensor([[0.9988, 0.8588, 0.1707, 0.0166, 0.0258]]),
       'stars': 1,
       'raw': tensor([[ 6.6865,  1.8058, -1.5807, -4.0844, -3.6306]])}

[11]: predict('There are some good things and bad things about this business')

[11]: {'proportional': tensor([[0.2590, 0.3784, 0.3179, 0.0350, 0.0096]]),
       'sigmoid': tensor([[0.6652, 0.9718, 0.8163, 0.0899, 0.0248]]),
       'stars': 2,
       'raw': tensor([[ 0.6864,  3.5408,  1.4915, -2.3147, -3.6728]])}

[12]: predict('There are some bad things and good things about this business')

[12]: {'proportional': tensor([[0.1187, 0.2201, 0.2986, 0.1887, 0.1739]]),
       'sigmoid': tensor([[0.2934, 0.5441, 0.7381, 0.4666, 0.4299]]),
       'stars': 3,
       'raw': tensor([[-0.8791,  0.1769,  1.0361, -0.1338, -0.2821]])}

[13]: predict('This is quite good. There are better products, but this is worth my recommendation too.')

[13]: {'proportional': tensor([[0.0057, 0.0314, 0.3342, 0.3823, 0.2464]]),
       'sigmoid': tensor([[0.0149, 0.0816, 0.8694, 0.9944, 0.6410]]),
       'stars': 4,
       'raw': tensor([[-4.1894, -2.4214,  1.8958,  5.1725,  0.5797]])}
```

REST API

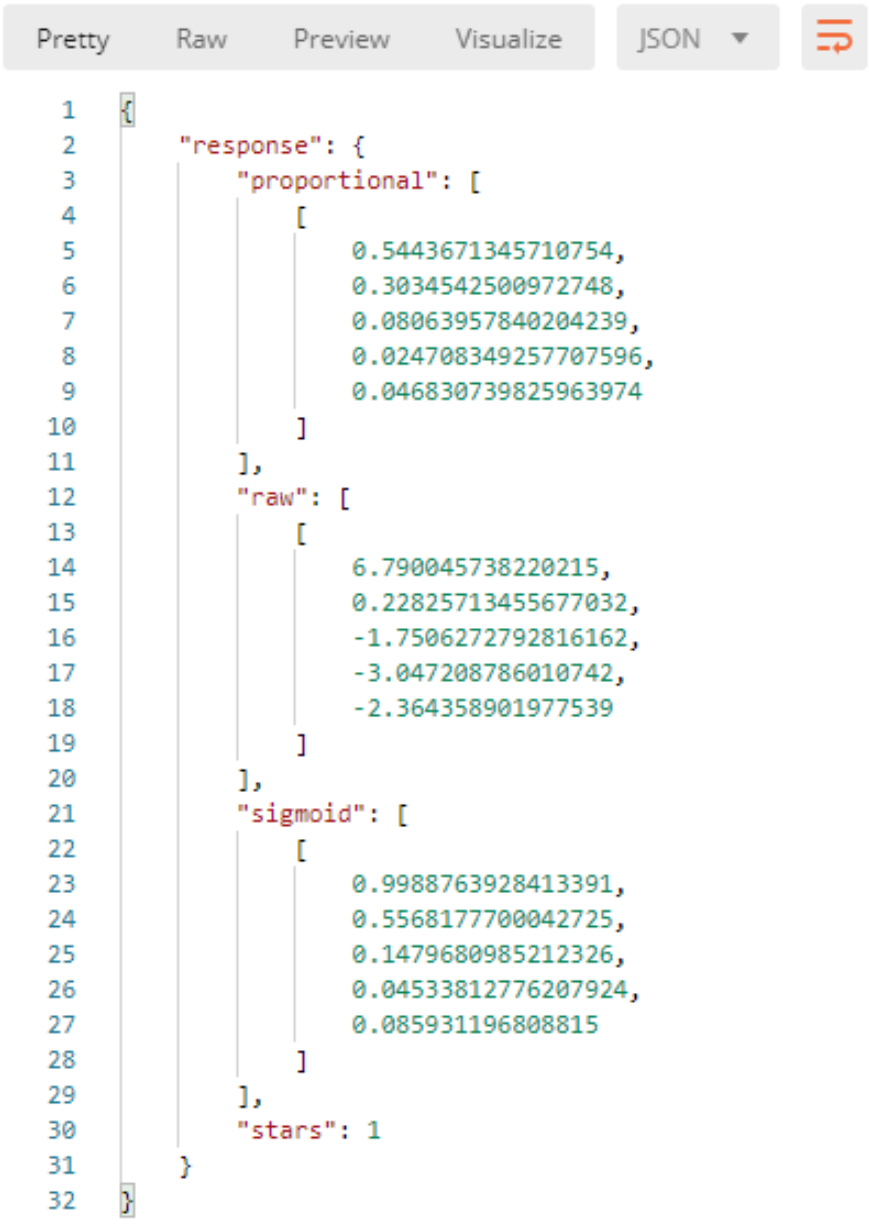
As the last and final step of our exercise we have created a REST API with Flask. The implementation of this API can be found here. The code in this script just re-uses the predict method from the prediction notebook and wraps it around with some Flask based code:

```
app = Flask(__name__)
```

```python
@app.route("/predict")
def predict():
    sentence = request.args.get("sentence")
    response = {}
    response["response"] = predict_sentiment(sentence)
    return flask.jsonify(response)


if __name__ == "__main__":
    app.run()
```

If you try for example the sentence from CNN below you will get one star rating with our model:

Boris Johnson's dream of a 'Global Britain' is turning into a nightmare



Conclusion

Hugging Face Transformers has all the material you need to create customized sentiment analysis models. Whilst very powerful, the BERT model is heavy when it comes to training models. Heavy in terms of GPU memory and also slow, but the results are satisfying. After trying out many sentences the star rating make some sense and when integrated into software can be really useful. Also the achieved accuracy of 71% on five stars is a good start.

The Yelp dataset would also offer you the possibility for creating some other very interesting models that can predict other categories like e.g: "funny", "cool" or even "useful". We have not explored these possibilities in this blog, but might do so in a future blog.

We will try to improve our experiment also by including other metrics and using parallelization with more data and describe our results in an upcoming blog.