Chapter 10

Modeling for NLP

# Main types of NLP challenges: sentiment analysis, question answering

Natural language processing (NLP) is a field operating on the intersection of linguistics, computer science and AI; its primary focus is algorithms to process and analyze large amounts of natural language data. Over the last few years, it has become an increasingly popular topic of Kaggle competitions. While the domain itself is very broad and encompasses very popular topics such as chatbots or machine translation, Kaggle contests dedicated to NLP problems usually focus on one of the following:

* Text classification (with sentiment analysis or intent classification as special cases)
* Open domain question answering
* Speech recognition

In this chapter we demonstrate some techniques focused on the first two problems: like in the chapter on computer vision, we demonstrate an example pipeline to handle an NLP problem. We conclude with a section on augmentation for NLP problems, which is a topic receiving significantly less attention than the vision counterpart.

# Baseline (shallow) methods

Before engaging in modeling with advanced techniques (like Transformer-based models for NLP), it is frequently a good idea to establish a baseline with simpler methods. To demonstrate the value of such approach, we will use the data from the Google Quest Q&A Labeling competition <https://www.kaggle.com/c/google-quest-challenge/overview/description> . The idea in this competition was to predict a set of labels for questions rated by humans on several diverse criteria – since the labels were aggregated across multiple raters, the objective was effectively a multivariate regression output, with target columns normalized to the unit range.

We begin by defining several helper functions, which can help us extract different aspects of the text.



The competition used in the metric was Spearman correlation (linear correlation computed on ranks <https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient>). Since we intend to build a scikit-learn pipeline, it is useful to define the metric as a scorer:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Part of the feature set we will use are embeddings from pretrained models – recall that the idea of this section is construction of a baseline without training elaborate models, but this need not prevent us from existing ones.

Obraz zawierający tekst

Opis wygenerowany automatycznie

We can now proceed to load the data and specify our 30 target columns of interest – for a discussion of their meanings and interpretation, the reader is referred to <https://www.kaggle.com/c/google-quest-challenge/data>

Obraz zawierający tekst

Opis wygenerowany automatycznie

Proceed with feature engineering: start by counting the words in the title and body of the question, as well as the answer. This is a simple, yet surprisingly useful feature in many applications:

Obraz zawierający tekst

Opis wygenerowany automatycznie

A next feature from the simple-yet-useful in NLP tasks is lexical diversity: counting the proportion of unique words in a chunk of text

Obraz zawierający tekst

Opis wygenerowany automatycznie

When dealing with information sourced online, it is useful to examine the components of website addresses:

Obraz zawierający tekst

Opis wygenerowany automatycznie

When examining the intent / sentiment of a question-answer pair, it might be useful to look at shared words:

Obraz zawierający stół

Opis wygenerowany automatycznie

Lots of stopwords can tell us something about the style, and intent:

Obraz zawierający tekst

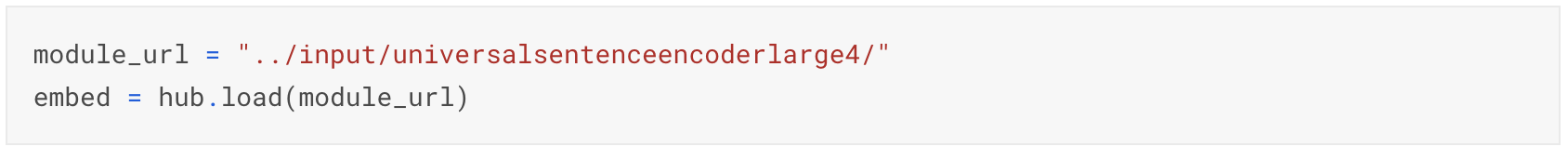
Opis wygenerowany automatycznie

When Terry Pratchett called multiple exclamation marks “a sure sign of a diseased mind” he may have been a bit on the harsh side, but keeping track of punctuation usage is potentially informative:

Obraz zawierający stół

Opis wygenerowany automatycznie

With the “vintage” features prepared, we can move to creating embeddings for the questions and answers alike. We could theoretically train a separate word2vec-type model on our data (or finetune an existing one), but for the sake of this presentation we will use a pre-trained model as-is. A useful choice is the Universal Sentence Encoder from Google <https://tfhub.dev/google/universal-sentence-encoder/4>. This model is trained on a variety of data sources, it takes as input a text in English and outputs a 512-dimensional vector.



The code for turning the text fields into embeddings is presented below:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Given the vector representations for both questions and answers, we can calculate the semantic similarity between the fields by using different distance metrics on the pairs of vectors:

Obraz zawierający stół

Opis wygenerowany automatycznie

Gather the distance features in separate columns:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Finally, we can also create a TF-IDF representations of the text fields – the general idea is to create multiple features, based on diverse transformations of the input text, and then feed them to a relatively simple model. We can achieve it by analyzing the text at word, as well as character level - in order to limit the memory consumption, we put an upper bound on the maximal number of both.



We instantiate char- and word-level vectorizers:

Obraz zawierający tekst

Opis wygenerowany automatycznie

The setup of our problem lends itself to a convenient usage of the Pipeline functionality from scikit-learn – one of the most fundamental toolkits for data wrangling:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

We wrap up the feature engineering part by processing the numerical features

Obraz zawierający tekst

Opis wygenerowany automatycznie

# Obraz zawierający tekst Opis wygenerowany automatycznie

A fast and efficient way to train our model to combine the preprocessing functionality with an estimator by using a pipeline:

# Obraz zawierający tekst Opis wygenerowany automatycznie

Once a model is fitted, we want to examine the performance. A convenient way to go about it is to create out of fold predictions. The procedure involves the following steps:

* Split the data into folds: in our case we use GroupKFold, since one question can have multiple answers (in separate rows of the data frame). In order to prevent information leakage, we want to ensure each question only appears within one fold.
* For each fold, train the model using the data of the other folds, generate the predictions for the fold of choice, as well as the test set
* Average the predictions on the test set

# Obraz zawierający tekst Opis wygenerowany automatycznie

We loop through the folds and build the separate models

# Obraz zawierający tekst Opis wygenerowany automatycznie

In this section, we demonstrate how to build descriptive features on a body of text. While not a winning formula for an NLP Kaggle competition, it is a useful tool to keep in one’s toolbox. Next, we move to the discussion of deep learning models, which have come to dominate the field in recent years.

# Sentiment analysis

Twitter is one of the most popular social media platforms and an important communication tool for many – individuals and companies alike. Capturing sentiment in the language is particularly important in the latter context: a positive tweet can go viral and spread the word, while a particularly negative one can be harmful. Since human language is complicated, it is important to not just decide on the sentiment, but also be able to investigate the “how”: which words actually led to the sentiment description? We will demonstrate an approach to this problem by using data from the Tweet Sentiment Extraction competition <https://www.kaggle.com/c/tweet-sentiment-extraction>.

We start by defining basic cleanup functions. First, we want to get rid of website urls, non-characters and replace the stars people use in place of swearwords with a single token “swear”.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Next, we remove html from the content of the tweets, as well as emojis:

Obraz zawierający tekst, stół

Opis wygenerowany automatycznie

Lastly, we want to be able to remove repeated characters (“waaaayyyyy” instead of “way”)

Obraz zawierający tekst

Opis wygenerowany automatycznie

For convenience, we ‘combine’ the four functions into a single cleanup one:



Last bit of preparation are functions for creating the embeddings based on a pre-trained model:

Obraz zawierający tekst

Opis wygenerowany automatycznie

And a combined tokenizer to work with a corpus:

Obraz zawierający tekst

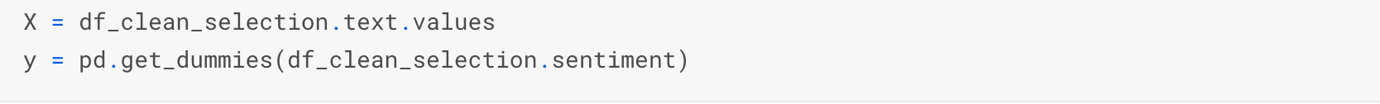
Opis wygenerowany automatycznie

After loading the data, it is usually a good idea to quickly inspect the format. In our case it is pretty straightforward:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Using our previously prepared functions, we can clean and prepare the training data. The sentiment column is our target, and we convert it to dummies for performance.



A necessary next step is tokenization of the input texts, as well as converting into sequences (along with padding, to ensure equal lengths across the dataset).

Obraz zawierający tekst

Opis wygenerowany automatycznie

We will create the embeddings for our model using DistilBert and use them as-is (like the approach in the preceding section, we could train the Embedding layer at gain performance – at the cost of massively increased training time). DistilBert is a lightweight version of BERT: the tradeoff is 3pct performance loss at 40pct fewer parameters.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Encode the tweets:

Obraz zawierający tekst

Opis wygenerowany automatycznie

With the data prepared, we can construct the model. For the sake of this demonstration, we will go with a fairly standard architecture for such applications: a combination of LSTM layers, normalized by Global Pooling and dropout, and a dense layer on top.

Obraz zawierający tekst

Opis wygenerowany automatycznie

There is no special need to pay attention to a temporal dimension of the data, so we are fine with a random split into training and validation – which can be achieved inside a call to the fit method:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Generating a prediction from the fitted model proceeds in a straightforward manner:

Obraz zawierający tekst

Opis wygenerowany automatycznie

In this section we demonstrate a sample pipeline for solving sentiment classification problems in NLP competitions. There are numerous improvements that need to be made if one is to achieve competitive performance (starting with larger embeddings), but the core elements are reusable:

* Data cleaning and preprocessing
* Creating text embeddings
* Incorporating recurrent layers and regularization in the target model architecture.

We can move to a second popular problem in NLP competitions on Kaggle: question answering.

# Question answering

In this section, we demonstrated an example pipeline for handling a frequent type of Kaggle NLP challenge: open domain question answering. We conclude by talking about augmentation strategies for text data.

# Augmentation strategies

Augmentation strategies for computer vision problems were discussed extensively in the preceding chapter. By contrast, similar approaches for textual data are less well explored topic (as evident by the fact there is no single package like albumentations). In this section, we demonstrate some of the possible approaches to alleviating the problem.

## Basic techniques

As usual, it is informative to examine the basic approaches first focusing on random changes and synonym handling. A systematic study of those is provided in Wei and Zou (2019) <https://arxiv.org/abs/1901.11196>.

We will use the data from the Tweet Sentiment Extraction competition mentioned earlier. First: synonym replacement.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

How would that work in practice?

Obraz zawierający tekst

Opis wygenerowany automatycznie

Not quite what you would call Shakespearean, but it does convey the same message while changing the style markedly. We can extend this approach by creating multiple new sentences per tweet:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Another simple strategy for improving generalization of our data is random deletion of words in sentences:

Obraz zawierający tekst

Opis wygenerowany automatycznie

How would it work in practice?

Obraz zawierający tekst

Opis wygenerowany automatycznie

Another simple approach is randomly swapping an order of words in the model – carefully applied, this can be viewed a potentially useful form of regularization, as it disturbs the sequential nature of the data that models like LSTM rely on.

Obraz zawierający tekst

Opis wygenerowany automatycznie

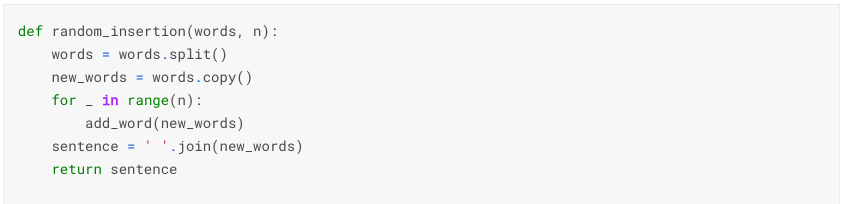
Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

Final one among simple methods is random insertion – this can be viewed as the NLP equivalent of adding noise or blur to an image.



Obraz zawierający tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst

Opis wygenerowany automatycznie

## Nouns and verbs

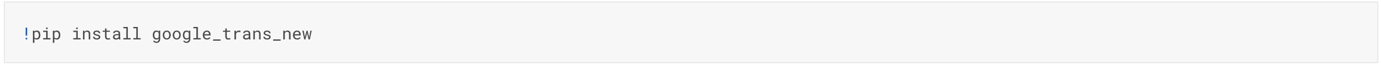
The augmentation methods in the previous section did not exploit the structure of the data to a high degree – even analyzing a simple characteristic like part of speech can help us construct more useful transformations of the original text. This is the approach we take in this section.

<https://www.kaggle.com/konradb/chap10-aug-nouns-and-verbs>

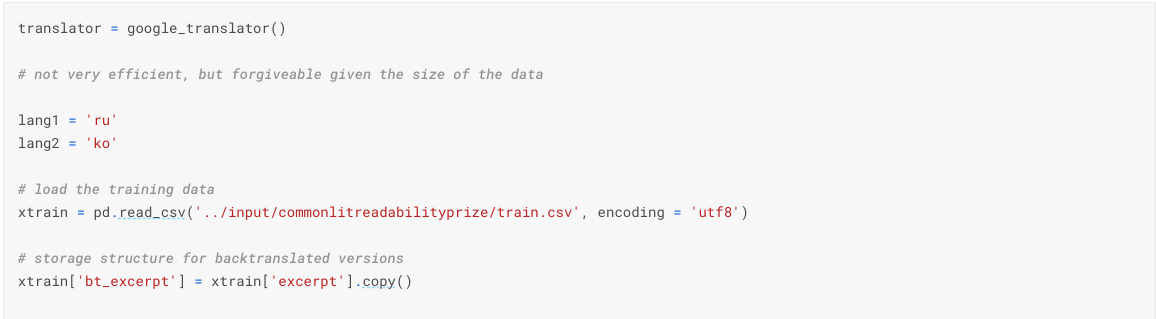
## Google translate

Anybody who has ever tried to use an online translator like Google Translate probably observed – especially if using one’s native language on either end – that while the gist of the message comes through, the algorithmic performance suffers when it comes to certain nuances of the next in a natural language. This opens a possibility of creating yet another augmentation technique: we translate from the source language to another one, and then back. Such a transformation (potentially utilizing multiple intermediate languages) yields a slightly distorted version of the original text. We will demonstrate this approach using data from the CommonLit Readability Prize competition <https://www.kaggle.com/c/commonlitreadabilityprize>

We begin by installing the Python interface to Google Translate – it is free of charge but comes with a limit on the number of API calls per unit of time, so for large scale applications there might be better options.



We take the original (English) text, translate it to Russian, then Korean, and then back to English.



Obraz zawierający tekst

Opis wygenerowany automatycznie

## Nlpaug

We conclude this section by demonstrating the capabilities provided by the nlpaug package: <https://github.com/makcedward/nlpaug>

<https://www.kaggle.com/konradb/chap10-nlpaug>

In this chapter we discussed the modeling for NLP competitions. We demonstrate both vintage and SOTA methods applicable to diverse problems such as sentiment analysis, question answering and text classification. In addition, we touched upon a frequently ignored topic – text augmentation.