How to summarise a set of caption lines to tell its topic?

Summarising a set of caption lines to tell its topic is a rather well-known problem, widely studied by many professionals throughout the past decade. Therefore, “text summarisation” tasks could now be accomplished with mere dozens lines of codes. The complex theories behind them, however, hold lasting spellbound, and the variation in approaches to the target is numerous. This brief survey will therefore try to describe the steps needed for a computer to learn the topic of a text.

* Categorise the model

In order to achieve the goal of summarising the topic of a piece of text, the model must take in a string of data that represents the text, and output the summery, either by choosing from a set of pre-set topics, or by directly generating the text. The latter is naturally more complicated and potentially less reliable, so the model investigated in this survey will be the former. As the model “makes a choice”, it falls under the category of a classification algorithm. As there will be multiple topics, the desired model should can be categorised into a “multi-class text classification model”.

* Choice of data

After a brief research on google, Reuters Newswire Topic Classification (Reuters-21578), a collection of news documents that appeared on Reuters in 1987 indexed by categories [1], appears to be the ideal choice. It is moderate in size, it consists of news captions, and its easy-access version on keras.datasets is heavily pre-processed, saving the time needed to write a new pre-processing function.

* Text-tokenisation

Text-tokenisation essentially transforms words (that a set of rules believes to be words) into other code-names (numbers), so that words are uniquely represented and can be fed into the model. The Reuters-21578 dataset stored in keras.datasets comes in tokenised in ‘count’ mode, where the word is replaced by the number of times it appeared in the text.

* Word embedding

Word embedding transfers words into a vector representing the word’s properties. This vector could be learnt by writing a separate algorithm (Word2Vec, GloVe, etc.), or it could be taken from learnt results in keras, as applied in the codes of this survey.

* Choice of algorithm

The three most popular machine learning algorithms in the field of NLP are Deep Neural Networks, 1D Convolutional Neural Networks and Recurrent Neural Network (mostly LSTM or GRU nowadays) All three are theoretically compatible with multi-class text classification, but they have their pros and cons.

Deep Neural Networks completely neglects the order of text. In sentiment analysis of text, a type of binary text classification problem, a Deep Neural Network might consider phrases with flattery compliments all negated by a negation a positive phrase. The same property probably makes it the worst choice in terms of accuracy.

1D Convolution and LSTM/GRU are better choices because they evade the downsides with a model with solely fully connected layers. In 1D CNN, words get to be considered as a group due to the presence of kernals and pooling layers. LSTM retains information which it finds important over long distances, so it could identify long-range relationships.

* Implementation

On the implementation level, I experimented with all three algorithms mentioned in the previous section. Deep Neural Network stagnates at around 80% validation accuracy and overfits onward. 1D CNN with word embedding requires more epochs to learn, but it steadily outperforms Deep Neural Network. Attempts to implement a model with LSTM with word embedding have unfortunately seen no success so far, with accuracy stagnant at around 44% at best due to unknown reasons.

[1]: https://machinelearningmastery.com/datasets-natural-language-processing/