The model

I choose to use a Convolutional Neural Network to do text classification. The ability of CNN in terms of text classification has been proved in a paper by Yoon Kim in 2014. Besides, I think it also has advantages on long documents because its features like pooling and sharing weights, which reduce the parameters in network as well as the computational cost.

We will usually use word embedding techniques such as word2vec or Glove to convert raw text into numerical data. However, the given text is hashed, which means we cannot use a pre-trained word embedding model directly. There are two methods to solve this problem. One is to train a custom word2vec model treating the hashed text as another language. The other is to build a dictionary, using the index of each hashed-word as numerical representation. I choose to use the second one. Because, training and tuning a word2vec model takes time. Besides, we can use a trainable Embedding layer which has the ability to learn word representations in our classification model.

We have to make documents have the same length before we feed them into the network. So, each document is zero-padded to the average length of documents.

The model is implemented in Keras using TensorFlow as backend. It has three conv layers with filter size [3, 4, 5], followed by pooling layers. Adam is used as optimizer and categorical crossentropy loss is chosen as loss function. I use an 8:2 train-test split. I believe the structure and performance of current network can be improved by adding batch normalization and applying attention mechanism. But, current network shows an 97% accuracy already after training with 5-fold cross validation.

The Flask App

After building the classification model. I tried to create a frontend which gives predictions after users enter their documents. I chose to use Flask, which is a micro web framework written in Python. Basically, a Flask App has the following structure: “app.py” gives the main logic of how does the App work. Files in “templates” folder define layout and components of webpage.

In my implementation, the App will load saved weights in order to initialize a classification model. It takes content which is entered by user as input. The input will be feed into the model to do prediction. The top 3 possible categories will be returned and showed on the webpage.

Training and Deploying Model on AWS Sagemaker

After the previous two steps, I tried to use services provided by AWS to train and deploy the model. I choose to use Sagemaker, which is a fully-managed platform that enables developers to build, train, and deploy machine learning models at any scale.

I implemented my model in Keras, which is not supported by Sagemaker currently. But, it provides us a way to use customized algorithms by wrapping our code in a docker image, and then push and build it on AWS. Data are stored in S3.

How will the model work is given by code located in “/container/text_classification”. “train” and “predict.py” are the most important two. The former defines what need to be done during model training, including model definition, data processing, etc. The latter defines how it will work when an endpoint of

the model is invoked. The model trained and deployed on Sagemaker shows lower performance since I do not train it as much time as it can due to the cost.

Sagemaker is powerful in terms of training and deploying machine learning model. But it does not provide a frontend. A serverless frontend can be built with the help of EC2 and S3, it can produce an API Gateway endpoint to trigger a Lambda function that will call the SageMaker endpoint.