INF5820, Fall 2018
Assignment 1: Neural Classification with Bags of Words

UiO Language Technology Group

Deadline 14 Sep., at 23:59, to be delivered in Devilry

Goals

1. Learn how to use the Abel HPC cluster to train deep learning models.
2. Get familiar with the *Keras* library
3. Learn how to use *Keras* to train and evaluate neural classifiers in NLP tasks.

Introduction

This is the first obligatory assignment in the INF5820 course, Fall 2018. It is comparatively small and aims at introducing you to the basics of neural network approaches to classification tasks in natural language processing. In it, you will implement a document classifier based on a simple feed-forward neural network, using bags-of-words as features. Primary and recommended machine learning framework in this course is *Keras* with *TensorFlow* as a backend. These are provided out-of-the-box in the LTG environment on the Abel cluster[1], and we strongly encourage you to train your models using Abel. If you would like to use any other deep learning software library, it is absolutely allowed, provided that you implement your own classifier, not simply take an off-the-shelf tool, and that your code is available.

Please make sure you read through the entire assignment before you start. Solutions must be submitted through Devilry[2] by 23:59 on September 14. Please upload a single PDF file with your answers. Your (heavily commented) code and data files should be available in a separate private repository in the UiO in-house Git platform[3]. Make the repository private, but allow full access to INF5820 teaching staff[4]. The PDF in Devilry should contain a link to your repository. If you have any questions, please raise an issue in the INF5820 Git repository[5] or email *inf5820-help@ifi.uio.no*. Make sure to take advantage of the group sessions on August 30 and September 6.

[1] `https://www.uio.no/studier/emner/matnat/ifi/INF5820/h18/setup.html`
[2] `https://devilry.ifi.uio.no/`
[3] `https://github.uio.no/`
[4] `https://github.uio.no/orgs/inf5820/people`
[5] `https://github.uio.no/inf5820/course2018/issues`

Recommended reading

1. **Neural network methods for natural language processing.** Goldberg, Y., 2017.[6]
2. **Speech and Language Processing**. Daniel Jurafsky and James Martin. 3rd edition draft of August 12, 2018. Chapter 7, 'Neural Networks and Neural Language Models' [7] (sections 7.1, 7.2, 7.3 and 7.4)
3. Linear Algebra cheat sheet by the LTG[8]
4. `https://keras.io/`
5. `https://www.tensorflow.org/`
6. `http://research.signalmedia.co/newsir16/signal-dataset.html`

1 Implementing a neural classifier

We will be working with the *The Signal Media One-Million News Articles Dataset*, which contains texts from news sites and blogs from September 2015. The dataset features several fields of metadata for each article, but in this assignment we will be interested only in one of them: the *source* of an article, i.e., from which news site it comes. This is a supervised classification task, since we have gold labels for this field. You'll have to implement a simple feed-forward neural network which takes bag-of-words (BOW) text representations as input and produces *source* predictions as an output, basically classifying the texts based on their source. The task includes processing the text data to extract BOW features, formatting these appropriately for the deep learning framework you are using (most probably, *Keras*) and experimenting with the classifier hyperparameters to come to an optimal solution.

The assignment is divided into six parts:

1. **Data processing**;
2. **Training a classifier**;
3. **Feature tuning**;
4. **Hyperparameter tuning**;
5. **Measuring time efficiency**;
6. **Evaluating the models**.

You have to submit a written report of 2-4 pages to Devilry, which provides details on your experiments and addresses the questions posed in the assignment. Your code should be available in your Git repository (we remind that the official programming language of the INF5820 course is *Python 3*). The code must be self-sufficient, meaning that it should be possible to run it directly on the data. If you use some additional data sources, these datasets should be put in the repository as well. If you use some non-standard *Python* modules, this should be documented in your PDF report.

[6] `https://www.morganclaypool.com/doi/10.2200/S00762ED1V01Y201703HLT037`
[7] `https://web.stanford.edu/~jurafsky/slp3/7.pdf`
[8] `https://www.uio.no/studier/emner/matnat/ifi/INF5820/h18/timeplan/la.pdf`

1.1 Data processing

We use the *The Signal Media One-Million News Articles Dataset*. In this assignment, you'll have to work with a subset of it, containing articles from 10 news web sites (they are represented best in the main dataset):

1. MyInforms
2. Individual.com
3. 4 Traders
4. NewsR.in
5. Reuters
6. Mail Online UK
7. App.ViralNewsChart.com
8. Latest Nigerian News.com
9. Yahoo! Finance
10. Town Hall

These sites are different in their dominant topics and vision; thus, we expect that the words used in their news articles would reflect these differences. Overall, this corpus contains about 55 thousand articles and about 12 million word tokens.

The prepared dataset is available on the UiO Github[9]. It is provided as a gzipped tab-separated text file, with one line corresponding to one news article. Data columns:

- **source** (class)
- **text**

The texts are already lemmatized and POS-tagged, and stop words are removed, to make NLP tasks easier. Thus, a sentence '*Satellite will play an important role in the fight against illegal fishing*' is converted to '*satellite_NN play_VB important_JJ role_NN fight_NN illegal_JJ fishing_NN*'. Note that the *Penn Treebank* tag set was used for part of speech tagging[10]. The documents are shuffled, so there is no inherent order in the dataset.

We want to train a classifier which will be able to predict the source of a news text based on the words used in it. For this, we need to somehow move from documents as character sequences to documents as feature vectors that can be fed to a machine learning algorithm, such as a feed-forward neural network. The obvious way to do this is to represent all the documents as bags-of-words: that is, extract the collection vocabulary (the union of all word types in the texts) and count frequency of each word type in each document (or mark the binary value of whether the word type appears in the document or not). Documents are then represented as sparse vectors of the size equal to the size of the vocabulary.

[9] `https://github.uio.no/inf5820/course2018/blob/master/obligatories/1/signal_10_5820.tsv.gz`

[10] `https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`

Machine learning classifiers can be trained on such data, which is what you are going to do.

We provide the dataset as is, but before training you have to split it into **train**, **development** and **test** parts, in the proportion of (80/10/10). You can also use the *Keras* integrated option to automatically extract the development/validation set. Make sure to keep the distribution of source labels as similar as possible in all 3 subsets. Describe the process of splitting the dataset in your PDF report in detail.

1.2 Training a classifier

We formulate our classification task as follows: given a list of words in a document, **predict the source** of this document.

You must write the code which takes the provided dataset and:

- infers a common dictionary for all documents in the dataset;
- extracts bags of words for each document;
- extracts a list of gold source labels for the documents (for example, `sources[i]` contains the source label for the i_{th} document);
- all this serves to prepare the data to be fed into a classifier.

Briefly describe this code in the report.

Then implement a fully-connected **feed-forward neural network** (multilayer perceptron) which trains on these bag-of-words features and evaluate it on the development dataset (see the '*Evaluation*' subsection below). For a start, you can look at the classifier code discussed at the group session [11]. Describe the parameters of your network and the evaluation results in the PDF report.

1.3 Feature tuning

Try to experiment with feature tuning. Introduce at least 3 variations of your BOW features. In order to do so you may glance at the lecture slides or at the literature for inspiration. Write a short description of your experiments. Provide examples.

1.4 Hyperparameter tuning

Experiment with the following hyperparameters of your neural network:

1. **Activation functions** (non-linearities);
2. **losses**;
3. **regularization terms**;
4. **optimizers**.

You should try at least 2 different values for each of these items and evaluate the resulting models on the development dataset. Report the results in your PDF report.

[11] `https://github.uio.no/inf5820/course2018/tree/master/neural_bow`

1.5 Time efficiency

For one hyperparameter, you should try at least 5 different values. This is the **number of hidden layers** in your neural network (starting from 1). For each number of layers, calculate the performance of the resulting model **and** the time it took to train this model. Draw a plot showing how performance and training time depend on the number of layers. This plot should be included in the PDF report, and the code you used to produce it must be committed to your repository.

1.6 Evaluation

It goes without saying that *all tuning of model hyperparameters must be done on the development set*, and the performance evaluations mentioned above should also use the same dev data. Note that you should report the following performance metrics for each model:

1. accuracy;
2. precision, recall and macro F1 score.

You can calculate these score using either *Keras*, *scikit-learn*, or any other mainstream machine learning library.

Describe the differences in your systems' performance in the PDF report. Give your opinion on the influence of different hyper-parameter values. Finally, **choose one best-performing model and evaluate it on the test dataset**.

Recall that non-linear models sometimes can produce different results with the same hyperparameters because of different random initializations. Thus, train your best model 3 times and evaluate it 3 times, providing the average and the standard deviation.

What are the results? Are they worse or better than on the dev data? Why? Report the results both in the PDF report and in a machine-readable format in your repository.

Good luck and happy coding!