Assignment 2: Sentiment Classification

Due Tuesday, April 23, 2020 at 11:59pm
TA in charge: Chan Young Park (chanyoun@cs.cmu.edu)

Collaboration Policy

You are allowed to discuss the assignment with other students and collaborate on developing algorithms at a high level. However, your writeup and all of the code you submit must be entirely your own.

1 Background

Sentiment classification, detecting if a piece of text is positive or negative, is a common NLP task that is useful for understanding feedback in product reviews, user's opinions, etc. Sentiment can be expressed in natural language in both trivial and non-trivial ways. Sometimes sentiment lexicon words such as "Good" and "Bad" are explicitly mentioned in the text and make the task easier. However, very often it could be much more subtle than that. For example, sentences like "but I'd rather poke out my eyes with a burnt stick" or "in fact after watching the rest of the season it has grown on me" do not contain any lexicon words, but as a human, we can tell those are conveying some sentiment. Negations, fixed expressions, idioms and many more can make this task quite tricky, and a simple lexicon-matching method simply won't work.

In this assignment, you will build two sentiment classifiers based on naïve Bayes and neural networks. Your models will be trained and evaluated on the provided movie reviews. We will use the percentage of correct predictions as our evaluation metric.

2 Files

Download the handout from Piazza, and unzip the archive. You should have the following files:

```
hw2-handout
├── dev_text.txt
├── dev_label.txt
└── heldout_text.txt
```

`dev_text.txt` contains reviews, one per line, and `dev_label.txt` contains their labels. You may use these files for training and testing your model, by splitting them. `heldout_text.txt` contains reviews that your models will be evaluated on. We purposely do not provide their labels, and you are not allowed to compare the true labels with your results for this assignment.

3 Task 1. Naïve Bayes (20 points)

Implement and train a naïve Bayes classifier on `dev_text.txt` and `dev_label.txt`. Run the trained model on `heldout_text.txt` and generate output file `heldout_pred_nb.txt` that contains predicted labels in the

same format as `dev_label.txt`. More specifically, we expect your source code file `naivebayes.py` to take four input arguments as in:

```
$ python3 naivebayes.py dev_text.txt dev_label.txt heldout_text.txt heldout_pred_nb.txt
```

The first three arguments are the provided data files, and the last argument is the output file name, in which the predicted labels will be written.

4 Task 2. Neural Networks (50 points)

Implement and train a neural classifier on `dev_text.txt` and `dev_label.txt`. Run the trained model on `heldout_text.txt` and generate output file `heldout_pred_nn.txt` that contains predicted labels in the same format as `dev_label.txt`. More specifically, we expect your source code file `neuralnet.py` to take the same four input arguments as in Task 1. We will not run this code on Gradescope, so you *must* submit `heldout_pred_nn.txt`.

Unless you are already familiar with other packages, we recommend you to use PyTorch for the implementation. The appendix includes instructions for the installation and the implementation.

You are allowed to

- Use basic building blocks provided by the package you use, such as RNN cells, optimizers, loss functions, etc.
- Use pre-trained word embeddings.

You are NOT allowed to

- Train a model on additional data that has sentiment-related labels.
- Use a model pre-trained on sentiment-related data.
- Copy-and-paste someone else's implementation.

5 Write-up (30 points)

For this assignment, in addition to submitting your source codes, you should turn in a write-up of the work you've done as `writeup.pdf`. The write-up should be 2-4 pages in length and should be clearly describe your code, model, and design choices you made. We do care about the clarity, so please keep your write-up as succinct as possible.

5.1 Naïve Bayes

For your implemented naïve bayes model, describe:

- How did you use the dev data (e.g., how to split them for training and testing)?
- How did you preprocess the data?
- For each feature you tried,
  - What is the feature (e.g., unigrams, bigrams, etc.; briefly in no more than 2 sentences)?
  - Why do you think it makes sense to use this feature?
  - Did the feature improve accuracy, either on the heldout or on your own test set?
  - Why do you think it did or did not improve accuracy?
- What is the final accuracy on your own test set?

5.2 Neural Networks

For your neural networks model, describe:

- How did you use the dev data, if different from Task 1?
- How did you preprocess the data, if different from Task 1?
- What is the architecture of your model? (recommended to include a figure that describes the entire architecture of your model)
- What does the model take as input?
- What features did you try other than word unigrams, if any? Did they improve accuracy?
- Any non-trivial extensions you made to the model
- What is the final accuracy on your own test set and how is it compared to your naïve Bayes classifier?
- Qualitative analysis on the model outputs and compare it with ones from your naïve Bayes

6 Final Submission

Compress the following files as `submit.zip` and submit it to Gradescope:

1. `writeup.pdf`
2. `naivebayes.py`
3. `neuralnet.py`
4. `heldout_pred_nn.txt`
5. `requirements.txt` (optional)

We will run your naïve Bayes classifier on Gradescope and use its output for evaluation. If you need to install external packages other than *numpy*, include those into `requirements.txt`. `pip3 install -r requirements.txt` will be executed before running `naivebayes.py`.

Note that you need to compress four files as one zip file, and not supposed to have a parent directory. In other words, if we unzip the `submit.zip`, there should be four extracted files, not a directory.

If you have consulted with other students or external resources, you *must* cite it in a separate section in `writeup.pdf`.

7 Appendix

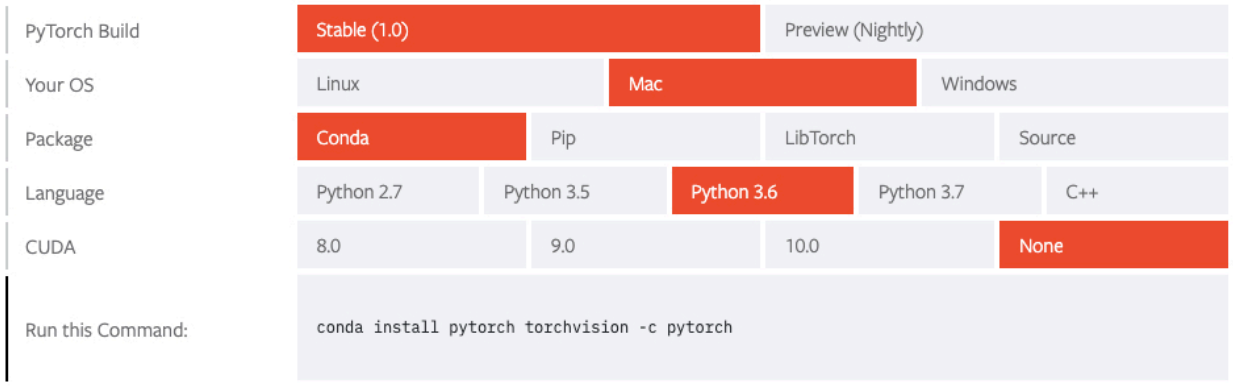
7.1 Installation

7.1.1 Virtual Environment

It is often convenient to have a separate virtual environment for a project. There are a variety of options, such as pipenv, virtualenv, and conda. Refer to the following article for the installation: https://medium.com/@krishnaregmi/pipenv-vs-virtualenv-vs-conda-environment-3dde3f6869ed

7.1.2 PyTorch

1. Go to https://pytorch.org/
2. In the middle of the page, follow the instructions under "QUICK START LOCALLY".
3. Your personal computer is most likely to have no GPUs, in which case you should choose None for CUDA.
4. Run the command shown at the bottom on your terminal app.



7.2 Implementation

The following tutorial shows step-by-step instructions for building a sentiment analyzer using a recurrent neural network (RNN).

https://github.com/bentrevett/pytorch-sentiment-analysis/blob/master/1%20-%20Simple%20Sentiment%20Analysis.ipynb

Please use this tutorial for guidance only. Make sure you use the dataset we provided, NOT as indicated in the tutorial.

7.2.1 Tips

- Check if your model learns anything, by looking at train and test losses over time.
- Keep track of input/output tensors' dimensionalities for individual function calls. `tensor.size()` is useful.
- Consider using pre-trained word embeddings, attention, and dropout to improve accuracy.

7.2.2 Useful Resources

- Official PyTorch documentation: https://pytorch.org/docs/stable
- Tensor basics: https://www.youtube.com/watch?v=jexkKugTg04, https://www.youtube.com/watch?v=fCVuiW9AFzY
- Simple network implementation (step-by-step): https://www.youtube.com/watch?v=_H3aw6wkCv0 (0:00:00-0:40:00)
- Book "Neural Network Methods for Natural Language Processing" for various concepts: https://www.morganclaypool.com/doi/abs/10.2200/S00762ED1V01Y201703HLT037
- How to debug neural networks: https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual-dc2a200f10f2
- Pre-trained word embeddings
  - GloVe: https://nlp.stanford.edu/projects/glove/
  - word2vec: https://code.google.com/archive/p/word2vec/
  - fastText: https://fasttext.cc/docs/en/english-vectors.html