CS 287
Assignment 1: Text Classification

In class we covered the basics of text classification, now it is time to get your hands dirty and implement it. In this homework assignment you will replicate current research baseline results for several text classfication tasks. Before starting the assignment, be sure that you understand all models described in the slides, in the text, and that you are familiar with the computational tutorials in HDF5 and Torch.

In this repo are several different data sets containing sentences and their classes. Begin by looking through the `data/` directory to get a sense of the different files. The main one we will use is called the Stanford Sentiment dataset (**?**).

```
> head data/stsa.fine.train
4 a stirring , funny and finally transporting re-imagining of
beauty and the beast and 1930s horror films
1 apparently reassembled from the cutting-room floor of any
given daytime soap .
1 they presume their audience wo n't sit still for a sociology
lesson , however entertainingly presented , so they trot out
the conventional science-fiction elements of bug-eyed monsters
and futuristic women in skimpy clothes .
2 the entire movie is filled with deja vu moments .
```

Here the first number gives the correct output class, and the rest of the line gives the input sentence. There is a similar file for the validation set. For the test set, we provide only the input.

As you complete this assignment, we ask that you submit your results on the test data to the Kaggle competition website at `https://inclass.kaggle.com/c/cs287-hw1` and that you compile your experiences in a write-up based on the template at `https://github.com/cs287/hw_template`. There are several other data sets in the repo that you may find useful for the write-up. For the Kaggle competition, you should submit your results for just the test set of the SST1 multiclass dataset.

1 Preprocessing

The first step is to convert the text representations of the data into features. For this assignment we have provided sample preprocessing code in `preprocessing.py`. This code provides preprocessing for all the data sets under `data/`. Please read the code carefully, as you will be expected to write preprocessing for future assignments.

The preprocessing code reads in text examples and writes them out as matrices containing the sparse features for each instance and the class. Our basic code just outputs unigram features from the data. The file will output a HDF5 bundle containing the following matrices: `train_input`, `train_output`, `valid_input`, `valid_output`, `test_input`, `nfeatures`, `nclasses`.

Note the structure of `X_input` and `X_output`. The input consists of features, where a single row contains the feature indices $f_1, \ldots, f_k$. However to account for differing sentence lengths, the row is padded with a special padding feature 1 to length k_{max} (these must be ignored in your modeling code). For this assignment, you should never need to explicitly construct the sparse vector $\boldsymbol{x}$. The output consists of the correct class index for each input. We do not provide the correct output for the test set.

2 Classification Setup

The main code for your project should be in `HW1.lua` and should be written in Lua/Torch. For this assignment, we ask that you construct all the models from scratch. You should not use any (non-debugging) libraries besides the standard Torch framework and HDF5.

2.1 Prediction and Evaluation

Be sure you are able to read and output the text data and that you understand the format. To confirm this, write a function that takes in the parameters of a linear model, and evaluates it on the validation set (`valid_input` and `valid_output`). The function should report the results in terms of prediction accuracy. This same code should be used for each of the linear models described in this project.

Also be sure you are able output your classification results on the test data as a text file. Check that you can upload these to the Kaggle.

2.2 Hyperparameters

Several of the models described have explicit hyperparameters that you will need to tune. It is your responsibility to cleanly separate these out from the models themselves and expose as command-line options. This makes it much easier to run experiments and to utilize experimental scripts.

2.3 Logging and Reporting

As part of the write-up, you will need to report on the training and predictive accuracy of your models. To make this possible, your code should report on various metrics of the model both at training and test time. We will leave it up to you on which metrics to log, but we recommend reporting training speed, training set loss, training set predictive accuracy, and validation predictive accuracy. It is your responsibility to convince us that the model is correctly training.

3 Models

For this assignment you will implement the three linear models that we discussed in class: naive bayes, multiclass logistic regression, and multiclass hinge loss.

3.1 Naive Bayes

We recommend that you implement multinomial naive Bayes first, since it is the most efficient and easiest to debug. Follow the description given in the class notes. Run and report on experiments for tuning the α hyperparameter.

3.2 Logistic Regression

Next implement multiclass logistic regression with L2 regularization (exposing the λ hyperparameter)[1]. For efficiency, training should be done using minibatch stochastic gradient descent. The size of the minibatch, the learning rate, and the number of epochs of training should also be exposed as hyperparameters.

Further notes:

- It is important to use the log-sum-exp trick discussed in the class notes. Without this, you will likely run into numerical issues.
- The Torch `index` and `indexAdd` operations should be helpful for improving efficiency.
- We recommend starting with a subset of the full training data to verify that your code is working. You can confirm this by tracking the loss of the training set.

3.3 Linear SVM

Finally implement hinge-loss with L2 regularization (exposing the λ hyperparameter). This code should use the same minibatch stochastic gradient descent code as Logistic Regression. The only difference is the calculation of the loss and the gradients.

3.4 Additional Experiments

Once these models are constructed, you should also report on additional experiments on these data sets. We will leave this aspect open-ended, but suggestion include:

- Implement 10-fold cross validation and experiment on other included datasets (see Wang and Manning (2012) or Kim (2014) for lists and results). Or find your own datasets to run online.
- Include additional features (for instance bigram features, suffix features, cluster features).
- Experiment with further models (for instance L2 loss SVM (Wang and Manning, 2012))

[1] Note that there are several great tutorials for how to implement logistic regression in Torch online. Do not use these. We expect the implementation to only utilize the core tensor operations in the framework. We will get to the some of the higher Torch abstractions in the next assignment.

- Extensive further hyperparameter tuning and experimentation (for instance Spearmint (Snoek et al., 2012))

4 Report and Submission

For your write-up, follow the report template at `https://github.com/cs287/hw_template`. Be sure to include a link to your code, Kaggle ID, and reports on your results.

In addition to submitting your Kaggle results, we also expect you to report on your experimental process. This should include data tables, graphs and discussion of any issues that you may run into.

References

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1746–1751.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959.

Wang, S. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics.