Natural Language Processing
Spring 2019

Assignment 2 (10 points)
Sentiment Analysis with Naïve Bayes and Recurrent Neural Networks

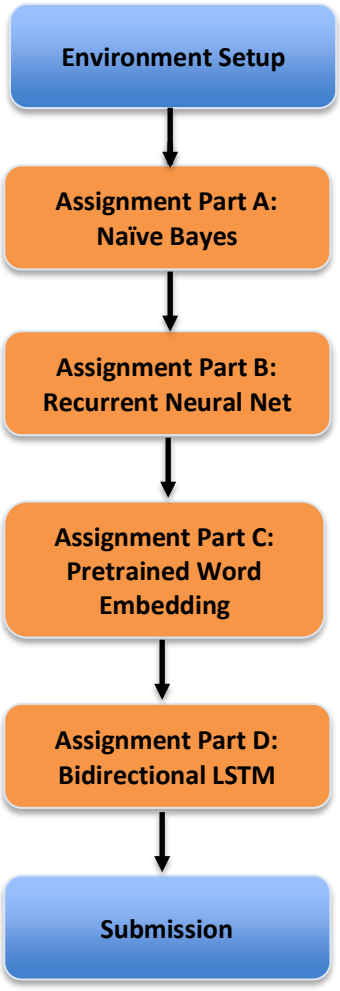
Due Friday, February 22, 11:59:59 PM

Introduction

In this assignment we will:

1. Train and Test a Naïve Bayes Classifier for Sentiment Analysis with Scikit-Learn on the IMDb movie dataset
2. Use Pytorch and Torchtext to develop a Recurrent Neural Network (RNN) Classifier for the same dataset
3. Use pre-trained word embeddings to improve the performance of the RNN
4. Convert the RNN to a bidirectional Long-Short-Term-Memory (LSTM) model

Note that it is entirely possible and likely to get variable (and potentially better!) results with different implementations of the code.
This document is structured in the following sequence:



Environment Setup

Once you have sshed into the Zoo cluster, copy the homework files to your hidden directory under Homework1 folder.

```
cp -r /home/classes/cs477/assignments/Homework2 ~/hidden/<YOUR_PIN>
```

For the virtualenv, all you need to do is just include a line in the environment setup section like:
Source the following environment to ensure that you are using the same packages that will be used for grading:

```
source /home/classes/cs477/venvs/hw2/bin/activate
```

If you wish to reproduce the virtualenv on your own computer, do the following:
While in the sourced virtualenv:

```
pip freeze > requirements.txt
```

Copy the requirements file to your computer

Then, create a virtualenv on your computer

```
virtualenv --python=python3 hw2
source hw2/bin/activate

pip install -r requirements.txt
```

Provided Files and Report

Report

Before starting the assignment, create a "README.txt" file on the homework folder. At the top, include a header that contains your NetID and name. Throughout the assignment, you will be asked to include specific output or comment on specific aspects of your work. We recommend filling the README file as you go through the assignment, as opposed to starting the report afterwards.

In this report it is not necessary to include introductions and/or explanations, other than the ones explicitly requested throughout the assignment.

Part A: Naïve Bayes

Time estimate to run correct implementation: 5 minutes

In this part of the assignment, you'll be training a Naïve Bayes Classifier on the IMDb Movie Review Database. You'll be using some of the features that come built into the sklearn library to quickly get a Multinomial Naïve Bayes Classifier running. Make sure to be filling out the template in the solutionsA.py file.

1) To process the movie reviews, we'll want to convert them into a matrix of word counts for each possible word. We recommend that you check out the CountVectorizer class in sklearn. You can use the nltk word tokenizer as well. Your code for this function should be very brief!
2) Now we'll train and test a Naïve Bayes Classifier on our dataset. Using sklearn's MultinomialNB() class, fit a classifier to the reviews and labels of the training set (x_train and y_train). Then return a set of predictions for the test set of reviews (x_test)

After completing Questions 1 and 2, enter the final accuracy of your model into your README file.

The result we got with a correct implementation was:
Accuracy: 67.69%

In addition to including the accuracy of the model in your README file, also include an explanation of the main assumption that is built into any Naïve Bayes classifier and how it is at work in this specific use-case.

Part B: Recurrent Neural Network

Time estimate to run correct implementation: 1 hour

This part of the assignment will involve building your own Recurrent Neural Network model for the same sentiment analysis task as earlier. We'll see that a simple RNN doesn't do better than the Naïve Bayes classifier, but we'll add some additional tricks to improve the performance.

1) The first thing you'll want to do is fill out the code in the initialization of the RNN class. You'll need to define three layers: self.embedding, self.rnn, and self.fc. Use the built-in functions in torch.nn to accomplish this (remember that a fully-connected layer is also a linear layer!) and pay attention to what each dimensions each layer should have for its input and output.
2) The next step (still in the RNN class) is to implement the forward pass. Make use of the layers you defined above to create embedded, hidden, and output vectors for a given input x.
3) The next function is the train function. Most of the code is handled for you- you only need to get a set of predictions for the current batch and then calculate the current loss and accuracy. For the latter two calculations, make sure to use the criterion and binary_accuracy functions you are given. For calculating the batch predictions, extract the text of the current batch and run it through the model, which is passed in as a parameter.
4) Your last step is to copy and paste what you did in step 3 into the evaluate function. This time, there's no additional optimization after the predictions, loss, and accuracy are calculated.

Include the accuracy of your model in your README.

The result we got with a correct implementation was:
Test Acc: 53.31%

Part C: Pretrained Word Embeddings

Time estimate to run correct implementation: 1 hour

This part of the assignment will involve a simple modification to the RNN model you just trained and tested.

1) This part of the assignment is very short. You need to modify 2 lines in main and copy over all of the work you did in the solutionsB.py file.

   Take a look at where we built up our vocabulary with the line TEXT.build_vocab(train_data, max_size=25000). You need to add an argument here (some research into the build_vocab function will help). We want you to use the glove.6B.100d pretrained embedding here.

   The next step is to extract the pretrained embedding (just thinking about the nature of the TEXT object is sufficient here). We'll then copy this into the model and see if there's any accuracy improvement.

The result we got with a correct implementation was:
Test Acc: 49.46%

Along with the final accuracy of your trained model, include a brief explanation for why pretrained word embeddings can be useful in Natural Language Processing tasks.

Part D: Bidirectional LSTM Network

Time estimate to run correct implementation: 5-6 hours

The final step of this assignment is to modify your RNN with pretrained word embeddings into a bidirectional LSTM network. We'll see that this kind of model performs much better than our previous ones. NOTE: This part of the assignment will take a long time to train, on the order of 5-6 hours. Start early so you have time to finish!!

1) You'll be making changes to your model in the RNN Class. In the init class, for the rnn layer, use the nn.LSTM function and make sure you pass in the bidirectional argument. Also note that the fully connected layer now has to map from two hidden layer passes (forward and backward).
2) In the forward pass, not much changes from before, besides the addition of the cell. Also note that you'll have to concatenate the final forward hidden layer and the final backward hidden layer. If any of this is unclear, look up example of how nn.lstm works for clarification.
3) Fill out any of the remaining incomplete functions with code from parts B and C.

The result we got with a correct implementation was:
Test Acc: 86.63%

Along with the final accuracy of your trained model, explain why a bidirectional LSTM network offers advantages compared to a "vanilla" RNN in your README.

Submission

You should submit your assignment through your hidden directory on the Zoo server. Once you are done with the homework and have finalized the README.txt file, check once again that this is the path for your homework:

~/hidden/<YOUR_PIN>/Homework2/

Please follow this folder structure exactly; otherwise you may lose points for otherwise correct submissions!

~/hidden/<YOUR_PIN>/Homework2/solutionsA.py

~/hidden/<YOUR_PIN>/Homework2/solutionsB.py

~/hidden/<YOUR_PIN>/Homework2/solutionsC.py

~/hidden/<YOUR_PIN>/Homework2/solutionsD.py

~/hidden/<YOUR_PIN>/Homework2/README.txt

As a final step, run a script that will set up the permissions to your homework files, so we can access and run your code to grade it:

/home/classes/cs477/bash_files/hw2_set_permissions.sh <YOUR_PIN>

Make sure the command above runs without errors, and **do not make any changes or run the code again**. If you do run the code again or make any changes, you need to run the permissions script again. Submissions without the correct permissions may incur some grading penalty.