Homework 3: Sentiment Classification with Naive Bayes and Recurrent Neural Networks (100 points)

Zhou Yu, Spring 2022
COMS W4705: Natural Language Processing

Due 4/29/2022 at 11:59pm ET

Overview

Part 1: Sentiment Classification with Naive Bayes (100 points)

In this part, you will be performing sentiment analysis on movie reviews by classifying the reviews as positive or negative. Sentiment analysis is used to extract people's opinions on a variety of topics at many levels of granularity. Our goal, however, for this assignment is to look at an entire movie review and classify it as positive or negative. In this part, you will implement Naive Bayes classifiers for sentiment classification.

Part 2: Sentiment Classification with Recurrent Neural Networks (Bonus: 20 points)

This part will serve as your introduction to using neural networks, a powerful tool in NLP. You will be doing emotion classification, in which you assign one of several emotion labels to a piece of text. You will implement and apply different neural architectures to this problem using PyTorch and work through the math that defines them. The data you will use for this homework is taken from a CrowdFlower dataset[1] from which a subset was used in an experiment published on Microsoft Azure AI Gallery[2].

Part 1: Sentiment Classification with NB

You and Naïve Bayes

We will be using **Naïve Bayes**, following the pseudocode in Chapter 6 of Jurafsky and Martin, using Laplace smoothing. The classifier you make will use words as features, add

[1] `https://www.figure-eight.com/data-for-everyone/`
[2] `https://gallery.azure.ai/Experiment/Logistic-Regression-for-Text-Classification-Sentiment-Analysis-1`

the logarithmic probability scores for each token, and make binary decisions between positive and negative sentiment. You will implement the binary version of the Naïve Bayes with boolean features and explore the impact of stop-word filtering. Stop-word filtering helps to improve performance by removing common words like "this", "the", "a", "of", "it" from your train and test sets. A list of stop words you will be using is included in the starter code at `NB/data/english.stop`

Finally you will implement your own features and or heuristics to improve performance. To help understand some key concepts, algorithms, and strategies with this assignment, we have included the paper from Pang and Lee [3]. All the code and data you need for this part is in the `NB/` folder.

What is Naïve Bayes?

Naïve Bayes is a classification technique that uses Bayes' Theorem with an underlying assumption of independence among predictors. Naïve Bayes classifiers assume a features presence is unrelated to another feature. For example if an orange is considered to be an orange if it is orange, round, and 4 inches in diameter; regardless of if two of these features are related or even existent upon another feature. Bayes' Theorem is the following

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

where A is the class and B is the predictor. There are many examples online, I will leave that for outside of this document.

There are several pros and cons to Naïve Bayes.

Pros

- Easy and fast to predict classes of test data. Performs well with multi class prediction
- **If** the assumption of independence holds, a Naive Bayes classifier performs better than most of models like logistic regression and with less training data.

Cons

- If categorical variables have a category that is not observed well in training, then the model will assign a 0 or near zero probability and will be unable to make predictions. To solve this *Zero Frequency* problem, we can use a smoothing technique like Laplace estimation.
- Bayes is known to be a bad estimator
- Assumption of independence is usually rare within a data set
- Performs poorly on a set where more false positives are identified than true positives.

So naturally how can we improve Naïve Bayes? This will be part of the assignment ☺

[3] **Thumbs up? Sentiment Classification using Machine Learning Techniques**

Details and Tasks

With the IMDB data set from the original Pang and Lee paper (included in the starter code), train a Naïve Bayes classifier. This code is already set up to perform 10-fold cross validation training and testing. Cross validation is simply splitting the data into several sections and then training and testing the classifier repeatedly. Specifically, this means training on 9 folds and testing on a different held out set. The average across the 10 iterations are calculated. This prevents bias towards a particular partition.

When using movie reviews for training, the fact that the review is positive or negative to train and help compute the correct values, but when using the same review for testing, only use the label to compute the accuracy. The data that comes with cross validation sections that are defined in the file `NB/data/poldata.README.2.0`

Task 1: Implement Classifier

Implement a Naïve Bayes classifier training and test method and evaluate them using the provided cross-validation mechanism. While Wikipedia[4] has the critical information to understand Naïve Bayes, there are great tutorials available from James Brownlee[5].

Splitting Data For 10-fold Cross Validation

The data has been split for you across all documents providing 10 splits.

Adding Documents

With the data split, we need to perform a series of operations on each split (or set of data). Notice in the train method, for every document, we use the addDocument function provided a classification (pos, neg) and a set of words. This function's task is to add the information the document provides in our Naïve Bayes model. Information worth adding in a dictionary count (similar to the unigram count in assignment 1) might be: the frequency of the classifier, the frequency of words, the frequency of words per classifier, and the number of words per classifier. Also potentially worthwhile is a similar listing for unique words in each document (for binary Naïve Bayes).

Adding The Classifier

In order to predict sentiment (classification), we need to calculate the probabilities of each sentiment. This is done with the following formula where c represents the particular classification.

$$c_{NB} = \underset{c_j \in C}{argmax}\ P(c_j) \prod_{i \in words} P(w_i|c_j)$$

$P(c_j)$ represents the probability of that classification in all possible classifications (pos, neg). This is a simple probability of just the number of documents classified with this

[4] `https://en.wikipedia.org/wiki/Naive_Bayes_classifier`
[5] `https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/`

sentiment! So the formal equation follows,

$$P(c_j) = \frac{N_{c_j}}{N_{c_{doc}}}$$

This gives us the first part of the classification equation above. Now let's look at the summation of individual words of that sentiment classification.

The probability of a word given a sentiment can be formalized by this equation,

$$P(w_i \mid c) = \frac{count(w_i, c)}{\sum_{w \in V} count(w, c)}$$

where $count(w_i, c)$ considers the frequency of a word in a sentiment given all words in a sentiment. This should remind you of homework 1 regarding unigram counts in the corpus!

Note that it is easiest to prevent underflow by decrementing a -log function by each of these probabilities as they are seen.

Task 2: Evaluate Model

Now evaluate your model again with the stop words removed. The implementation of using the stop words has been provided and can be run by running the following command

```
python NaiveBayes.py -f /data/imdb
```

inside the folder `NB/`.

Task 3: Binary Version of Naïve Bayes

Now implement a binary version of Naïve Bayes classifier that uses the presence or absence of a feature rather than the feature counts. Let's look at a formal equation for a Binary Naïve Bayes,

$$c_{NB} = \underset{c_j \in C}{argmax}\ P(c_j) \prod_{k \in words} P(w_k \mid c_j)$$

where $P(w_k, c_j)$ is equal to the Binary event model,

$$P(w_k \mid c_j) = \begin{cases} \frac{doc\_count(w_k, c)}{\sum_{w \in V} doc\_count(w, c)} & \text{when} \quad doc\_count(w_k, c) > 0 \\ 0 & \text{when} \quad doc\_count(w_k, c) = 0 \end{cases}$$

where w_k represents each word in the vocabulary. To clarify, count in this case is the document frequency as opposed to the actual word frequency. **Do not confuse this with Bernoulli Naïve Bayes** which penalizes the probability when the vocabulary word for a sentiment does not exists in the document. This might be worthwhile to implement when creating your best model.

Task 4: Interpret the Model Behavior

Recall that Naive Bayes classifiers make predictions by

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_{i \in words} P(w_i|c_j)$$

We can interpret and understand the model behavior by analyzing the value of $\Pr(w_i|c_j))$ for each class c_j and word w_i. For each class c_j, what are the signal words w that are indicators of an example being class c_j? More concretely, for each class c_j, what are the words w that maximizes $\Pr[w_i|c_j] - \Pr[w_i|\neg c_j]$?

Fill in the `analyze_model` function in `NB/NaiveBayes.py` that takes a NB classifier as input, and outputs two word lists each consisting of 10 words corresponding to a class.

Task 5: Can Our Model Be Improved?

Experiment with different heuristics and features to increase the classifier's accuracy. Here are some pointers that should help you. We also recommend looking at the data and reading the suggested paper from Pang and Lee. **In order to get full credit, you need to experiment with at least two new heuristics and features.** Clearly document at the top of the `NaiveBayes.py` file the new heuristics and features you implemented and experimented with.

- **Adding More/Other features**: You'll have used the unigram words in each review as features in your model. What other features can you use that will boost your classifier's performance?
- **Feature Selection**: Feature selection is a tool that can be used to remove some possibly-redundant features. The stop word-removal is a simplistic feature selection, and there are more sophisticated ones from the reading. See if any of these feature selection techniques can be used here to give better cross-validation scores. One strategy, removing correlated features, is a good place to start as the highly correlated features are accounted for twice can lead to over inflating importance in the model.
- **Laplace Smoothing**: Laplace smoothing is a must for Naïve Bayes, and you should be doing it already in your model. However Bayes classifiers have limited options for parameter tuning. It is highly recommended to focus on pre-processing of data and the feature selection rather than specific parameter tuning. Are there other smoothing techniques that could help?
- **Weighting features**: In the multinomial Naïve Bayes classifier, each feature was weighed according to the frequency. The binary version was weighed by presence and absence. Perhaps there are some better ways to weigh the features?
- **Classifier Combination Techniques**: We would not recommend classifier combination techniques like bagging, boosting, and ensembling because their purpose is to reduce variance, but with Naïve Bates there is no variance to minimize.

This link has some great insights to improving Naïve Bayes performance[6].

There are also some common techniques to improve the performance of Naïve Bayes. Here are some examples of where improvements should be made.

- *Many people thought this movie was very good, but I found it bad.* This sentence has two strong and opposing words in this sentence (good, bad), but it can be inferred that the sentiment of the sentence can be determined by the word bad and not good. How can you weigh your features for 'good' and 'bad' differently to reflect this sentence's sentiment?
- *Paige's acting was neither realistic or inspired.* Currently, the feature set comprises individual words in a bag of words approach. Because of this, the words inspired and realistic are considered separately despite the surrounding words like neither. How can you model take into consideration this word order?
- *The sites are chosen carefully; it is no wonder this comes across as a very beautiful movie.* How can you change the weight for a feature like beautiful, since it is expressed more strongly than saying 'it is a beautiful film'. How does very impact the next word?

Some notes on generalization It is important to come up with simple features that maintain generality and are not fixed to specific examples. Try to choose features that model your both training and held-out/unseen data as opposed to just examples in the training set. **Optimizations regarding just the provided data set, will not do well in the held out set.**

1. Your Implementation

To ensure that your code functions properly from the command line and our grader, you should limit your changes to addDocument() and classify() prototypes in `NB/NaiveBayes.py`. Feel free to add other elements invoked by these methods. Note main() is not executed so you cannot rely on anything added there.

The grader calls the following functions: crossValidationSplits(self, trainDir), trainSplit(self, trainDir), train(self, split), and test(self, split). It also relies on the definitions of classes TrainSplit and Document with the associated flags. Note that for your best model, you should have the appropriate flags turned on including, for example, stopWordsFilter if your best model uses it.

2. Evaluation

Your classifier will be evaluated on two datasets: included IMDB and a second, held-out test set. This will be done for Naïve Bayes with and without stop-word filtering, binary Naïve Bayes, and your best model.

[6] `https://machinelearningmastery.com/better-naive-bayes/`

Minimum Requirements

1. Your best model should be achieve higher probability than your basic naive bayes classifier and the binary version naive bayes classifier.
2. It will need to achieve at least 83% average accuracy with the 10-fold cross validation on imdb dataset.
3. It will need to achieve a higher accuracy than a TA model on the hold out dataset.

Competitive Task Try to improve your classifier as much as you can! We will test all of your classifiers with a hold out dataset. The top 10% classifiers with the highest prediction accuracy will be awarded 5 bonus points for this assignment!

3. Running Your Classifier

Use the command line to run this code. You may have issues in IDEs like PyCharm because the NaiveBayes must be able to find the data directory in the default location. Your code will run with the command

```
$ python NaiveBayes.py data/imdb
```

Adding flags should be done as such. Flag -f adds stop word filtering, flag -b uses your binarized model, and flag -m invokes the best model. **These flags should be used one at a time**. Your assignment will be graded similarly using one flag at a time.

If you wish to use your classifier on other data sets, you can specify a second directory to test on the entirely held out second set! This can be done with the following command.

```
$ python NaiveBayes.py -fbm <train directory> <test directory>
```

Part 2: Sentiment Classification with RNN (Bonus)

To get started, download the provided code from the website, and the folder `RNN/` contains all the code and data for this part. Ultimately, the provided code and the code you write should work together to load the text data from the `RNN/data/crowdflower_data.csv` file, preprocess it and place it into Pytorch `DataLoaders`, create a number of models, train them on the training and development data, and test them on a blind test set. Most of the code is already written for you; you will complete this assignment by filling in some code of your own.

1. Provided Resources

The data in `RNN/data/crowdflower_data.csv` consists of tweets labeled with 4 emotion labels: 'neutral', 'happiness', 'worry', and 'sadness'. Each tweet has exactly one label. The data is not pre-processed in any way.

Code is already provided to 1) load, preprocess, and vectorize the data; 2) load pre-trained 100-dimensional GloVe embeddings; and 3) test a generic model on the test set. The preprocessing code is located in `RNN/utils.py`. You may **not** modify `test_model()`.

The `main()` function in `RNN/main.py` is provided to start you out; it loads and pre-processes the data, and will save it to file if you set `FRESH_START = True` and load it if you set `FRESH_START = False` so that you do not have to re-process the data every time you run the code. You should use this function to run and test your code.

2. Tasks

In this assignment you will need to do the following:

1. Fill in the `train_model()` function in `RNN/main.py` to train your models (§)
2. Implement a recurrent neural network
3. Fill in `main()` in `RNN/main.py` to run your models

3. Training Code

You will need to fill in the `train_model()` function in `RNN/main.py` to train your models. You may **not** modify the function header. The `train_model()` function you submit should do the following:

- Train by looping through minibatches of the whole training set;
- Calculate the loss on each minibatch (between the gold labels and your model output) using the existing loss function;
- Do backpropagation with the loss from each minibatch and take an optimizer step;
- At the end of each epoch, calculate and print the total loss on the development set;
- Train until the loss on the development set stops improving; and
- Return the trained model.

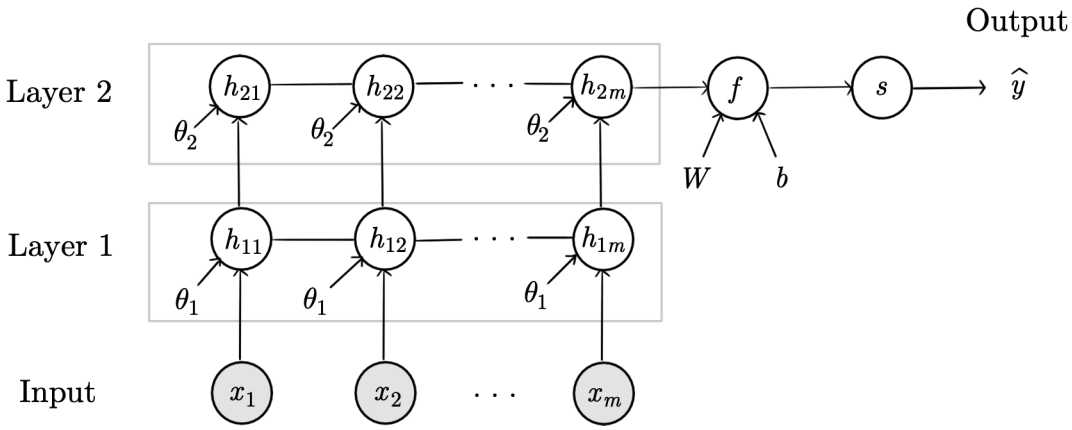
4. Models

You will implement a recurrent neural network, by filling in the `init()` and `forward()` functions for the `RecurrentNetwork` class in the `RNN/models.py` file.

Recurrent Network

Your recurrent network should follow the computation graph below:

You can choose the type of RNN (plain RNN, LSTM, GRU).



Deliverables and Submission Instructions

Submit **one** zip file named <**YOUR-UNI**>_**hw3.zip** to Gradescope. Your zip file should contain exactly the same files we released in the skeleton code, and you may not change the file structure of the skeleton folder. You should submit the same set of files even if you were not able to complete some part of the sections.

Don't forget to cite your sources as well!