Assignment 1 Report
CSE 143: Intro to Natural Language Processing

University of California Santa Cruz
Tommaso Framba, tframba@ucsc.edu, ID: 1815342

Tuesday April 26, 2022

This assignment was done in Python 3 without use of any NLP libraries (such as NLTK).

Abstract

In this first assignment I implemented a Naive Bayes classifier by hand (without programming) for sentiment analysis. As well, I implemented a Naive Bayes, and Logistic Regression classifier inside of the classifier.py skeleton code to provide sentiment analysis for Hate Speech text.

1 Problem: Naive Bayes by Hand

In this first problem, I implemented a Naive Bayes classifier by hand (without programming) for sentiment analysis. All scores are reported as **log-probabilities**, to three significant figures. A collection of movie reviews data was given and a sentence for classification for the fitted model was given below:

S:

```
The film was great, the plot was simply amazing! Makes other
superhero movies look terrible, this was not disappointing.
```

(a) Part a asked for training of the model to detect positive vs negative reviews and for the predicted scores $P(+|S)$ and $P(-|S)$) for the sentence S:

First I calculated the prior probabilities of 3 positive and 3 negative classes.

$$P(+) = 0.5$$
$$P(-) = 0.5$$

Then I calculated the log probability of each word in order to train the model.

+:	-:
$P(great\|+) = log(6/23) = -0.584$	$P(great\|-) = log(2/23) = -1.041$
$P(amazing\|+) = log(8/23) = -0.459$	$P(amazing\|-) = log(1/23) = -1.342$
$P(epic\|+) = log(5/23) = -0.663$	$P(epic\|-) = log(4/23) = -0.74$
$P(boring\|+) = log(2/23) = -1.061$	$P(boring\|-) = log(5/23) = -0.643$
$P(terrible\|+) = log(1/23) = -1.362$	$P(terrible\|-) = log(4/23) = -0.74$
$P(disappointing\|+) = log(1/23) = -1.362$	$P(disappointing\|-) = log(6/23) = -0.564$

I then classified the sentence S as follows:

$$P(+|S) = P(+) * 10^{sum(P(word|+))}$$
$$P(+|S) = 0.5 * 10^{((-0.584)*(-0.459)*(-1.362)*(-1.362))} = 0.0000855$$
$$P(-|S) = P(-) * 10^{sum(P(word|-))}$$
$$P(-|S) = 0.5 * 10^{((-1.041)*(-1.342)*(-0.74)*(-0.564))} = 0.0001027$$

Model will apply negative (-) class as $P(-|S) > P(+|S)$

(b) In part b I implemented the same technique with added add-1 smoothing. I accomplished this by adding (1) to the numerator and count of features per class (6) to the denominator. All other calculations were identical to part (a).

Calculate log probabilities

+:	-:
$P(great\|+) = log(7/29) = -0.617$	$P(great\|-) = log(3/28) = -0.97$
$P(amazing\|+) = log(9/29) = -0.508$	$P(amazing\|-) = log(2/28) = -1.146$
$P(epic\|+) = log(6/29) = -0.684$	$P(epic\|-) = log(5/28) = -0.748$
$P(boring\|+) = log(3/29) = -0.985$	$P(boring\|-) = log(6/28) = -0.669$
$P(terrible\|+) = log(2/29) = -1.161$	$P(terrible\|-) = log(5/28) = -0.748$
$P(disappointing\|+) = log(2/29) = -1.161$	$P(disappointing\|-) = log(7/28) = -0.602$

I then classified the sentence S as follows:

$$P(+|S) = P(+) * 10^{sum(P(word|+))}$$
$$P(+|S) = 0.5 * 10^{((-0.617)*(-0.508)*(-1.161)*(-1.161))} = 0.000179$$
$$P(-|S) = P(-) * 10^{sum(P(word|-))}$$
$$P(-|S) = 0.5 * 10^{((-0.97)*(-1.146)*(-0.748)*(-0.602))} = 0.000171$$

Model will apply positive (+) class as $P(+|S) > P(-|S)$

(c) Part c asked what are some additional features that you could extract from text to improve the classification of sentences like S?

You could extract bigrams, trigrams, or even n-grams from the text as additional features. This would allow you to rate bigrams like "Not disappointing" more positively. This would help improve the overall classifications of sentences like S because of words that by themselves as unigrams appear to be negative but when combined with previous words as bigrams, in reality, have a more positive connotation.

2 Programming: Hate speech detection

2.1 Naive Bayes

Model Description In this part, I implemented a Naive Bayes model with *add-1 smoothing*, as discussed in the section and in the readings. The `NaiveBayesClassifier` class in `classifiers.py` was fully implemented with proper init, fit, and predict methods. This model primarily relies on a method `wordprobabilities` that takes in the text and label numpy arrays and provides a probability for each word into two positive and negative class dictionaries. Before the word probabilities can be calculated the model calls the `priorprob` method to determine the prior probabilities of the individual classes. Finally, once the model is fitted the predict method enters a row by column loop that runs in O(rows * cols) and

determines the proper class by adding up the log probabilities and then exp'ing them. The final probabilities are then returned.

Performance Overall the performance of this model is not the best but still runs in O(row*cols) time due to the word probability and predicting methods. The model can be fully fitted in around twenty seconds on the `train.csv` file.

Experimental Procedure In order to implement the NaiveBayesClassifier class in the `classifiers.py` file I naturally began by completing problem 1 in this assignment. By completing problem 1 I was able to break down the necessary steps to implement the class. These steps needed were to calculate prior probability of classes, word probability per class, and predictions based on those probabilities. For these steps I researched the numpy library to see what methods I could use to obtain a correct result. The two methods I found most useful were the np.exp() and np.log() methods.

Deliverables

1. Report the accuracy on the training, dev, and test sets. Model was trained on `train.csv`.

$$TrainAccuracy : 1323/1413 = 0.9363$$
$$TestAccuracy : 184/250 = 0.7360$$
$$DevAccuracy : 176/250 = 0.7040$$

2. Below are reported observations on four randomly selected examples in the dev set. The first two are correct classifications and the next two are incorrect classifications.

Correct Rows: 227, 242
Incorrect Rows: 235, 210

Observations:
I observed that the incorrect rows do not have a bad context or connotation but since the words are being classified as unigram features it looks at the individual words such as white for row 235 and sees that it contains a race which by itself may have overall been classified as negative or hate speech. I also observed that the correct rows contain almost no words that have negative connotations and include nouns that are not races but things like sports such as in row 242.

3. Below are ten words with highest $P(w|1)/P(w|0)$ ratio and 10 words with the lowest ratio.

`Highest: Aids, Apes, Asian, Filth, Jews, Leave, Liberal, Mud, Non, Running`

`Lowest: Check, Father, Sf, Spirit, Sports, 15, [, ], _, Thanks`

Trends: From these ten words I can observe that the highest ratio of $P(w|1)/P(w|0)$ contain words that can be described as races or racial slurs. Some other words in the highest ratio such as aids can be described as diseases and words such as non or leave can be described as negations and overall carry a negative connotation by themselves. This is important because the model classified the features as unigrams. For the lowest $P(w|1)/P(w|0)$ ratio words I can see that the words are topics such as sports or delimiters.

2.2 Logistic Regression

Model Description In this part, I implemented the `LogisticRegressionClassifier` class in `classifiers.py` with and without L2 regularization on the train, dev, and test sets. The model utilizes stochastic gradient descent to arrive to a correct set of weights to classify text into hate or non hate speech.

Performance Overall the performance of this model is not very good and could be improved. As implemented it runs in O(epoch*row*cols) time due to the way that I implemented my stochastic gradient descent algorithm.

Experimental Procedure In order to implement the LogisticRegressionClassifier class in the `classifiers.py` file I naturally began by rewatching Yuja recordings of lecture and attending Brendan Kings section and office hours. By attending these lectures I was able to derive the formula that stochastic gradient descent undergoes for each iteration of an epoch. This was crucial in understanding the necessary steps to implement the class. These steps needed were to calculate error per step, updating the gradient of cost, and creating predictions based on fitted weights. For these steps I reiterate that understanding and breaking down lecture and section notes were crucial to the overall success of the implementation.

Deliverables

1. Report the accuracy on the training, dev, and test sets. Model was trained on `train.csv`.

$$TrainAccuracy : 1313/1413 = 0.9292$$
$$TestAccuracy : 175/250 = 0.7000$$
$$DevAccuracy : 181/250 = 0.7240$$

   Comparison to Naive Bayes: Overall the way that I derived my algorithm was by deconstructing a sctochastic gradient descent algorithm so I am not very surprised that the results are decent when compared to naive bayes. Although it is lesser in value for the train and test sets the accuracy for the dev set was better in non l2 regularization Logistic Regression. If I were to classify without l2 regularization I would use Naive Bayes instead because of the overall run time being far faster with Naive Bayes.

2. Add L2 regularization with different weights, such as $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

$$\text{Best results with } \lambda = 1, epoch = 40, \alpha = 0.1$$
$$TrainAccuracy : 1392/1413 = 0.9851$$
$$TestAccuracy : 181/250 = 0.7240$$
$$DevAccuracy : 185/250 = 0.7400$$

   Observations: Out of all the possible λ values I personally found that 1 yielded the best results when used with a mix of $epoch = 40$ and $\alpha = 0.1$. The other regularization values tended to undefit or overfit the data and yielded poor accuracy results. I think this is due to the way how I implemented my stochastic gradient descent by deriving how a stochastic gradient descent with l2 regularization performs. In the end I am incredibly pleased with my result because of how close it was to the sklearn libraries score values for built in Logistic Regression. Since I know that sklearn utilizes l2 regularization by default I believe that my algorithm provides a decent result for performing Logistic Regression with stochastic gradient descent and L2 regularization from scratch.