CS 4650 (Spring 2021) HW1 – NB & LR

January 2021

Instruction

1. This homework has two parts: Q1 and Q2 are theory problems, and Q3 is a coding problem with some parts requiring a written answer.
2. We will be using Gradescope to collect your assignments. Please carefully read the following instructions for submitting to Gradescope.
   - (a) Each subproblem must be submitted on a separate page. When submitting to Gradescope (under **HW1 Writing**), make sure to mark which page(s) correspond to each problem or subproblem.
   - (b) For the coding problem, please upload codes under the **HW1 Programming** assignment on Gradescope. In addition, include the writeup for each subproblem of this coding problem in your solution PDF.
   - (c) Note: this is a large class and Gradescope's assignment segmentation features are essential. Failure to follow these instructions may result in parts of your assignment not being graded. We will not entertain regrading requests for failure to follow instructions.
3. LaTeX solutions are strongly encouraged (solution template available on the class website), but scanned handwritten copies are also acceptable. Hard copies are not accepted.
4. We generally encourage collaboration with other students. You may discuss the questions and potential directions for solving them with another student. However, you need to write your own solutions and code separately, and not as a group activity. Please list the students you collaborated with on the submission site.

1 Naïve Bayes

A collection of movie reviews (data $\mathcal{D}$) contains the following keywords and binary labels for whether each review is positive (+) or negative (−). The data is shown below: for example, the cell at the intersection of "Review 1" and "epic" indicates that the text of Review 1 contains 2 tokens of the word "epic". Answer the following questions, reporting all scores as **log-probabilities**, to three significant figures.

Review	great	amazing	epic	boring	terrible	disappointing	Y
1	5	0	2	1	0	0	+
2	2	2	2	0	0	1	+
3	3	1	2	0	1	1	+
4	2	0	1	2	0	1	−
5	2	0	1	2	2	0	−
6	2	1	0	2	2	2	−

(a) Assume that you have trained a Naïve Bayes model on data $\mathcal{D}$ to detect positive vs. negative movie reviews. Compute the model's predicted scores for both positive and negative classes for the following sentence S (i.e. $P(+|S)$ and $P(-|S)$), and determine which label the model will apply to S. (4 points)

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

(b) The counts in the original data are sparse and may lead to overfitting, e.g. a strong prior on assigning the negative label to reviews that contain "terrible". Apply *add-1* smoothing and recompute the Naïve Bayes model's predicted scores for S. Did the predicted label change? (4 points)

(c) What is an additional feature that you could extract from text to improve the classification of sentences like S (not necessarily in NB), and how would it help improve the classification? (2 points)

2 Logistic Regression

2.1 Gradient Descent Updates

Logistic Regression is used to model the probability of a data point belonging to a class or an event occurring. For this, we apply the sigmoid function to a linear combination of independent variables. Logistic Regression is a versatile model that can be used for many different types of models.

Mathematically, logistic regression is defined as:

$$\hat{y} = \sigma(z)$$
$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$z = (wx + b)$$

The result is the probability that tells us how the model would classify the given data-point x based on the learnable weights w and bias b. Let's assume we have a labelled dataset for detecting hate speech. We can then use Gradient Descent to optimize the weights of the model. However, we need to give Gradient Descent a good loss function to optimize. Let's use the cross entropy loss function for this. Since the cross entropy loss is convex for logistic regression, it has only one minima.

$$\mathcal{L}_{CE}(\hat{y}, y) = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$

where $\hat{y}$ is the predicted probability and y is the ground-truth label.

With gradient descent,

$$w^{t+1} = w^t - \eta \frac{\partial \mathcal{L}}{\partial w}$$

where η is the learning rate.

(a) Calculate the partial derivative of the cross entropy loss with respect to the weights w and bias b. (4 points)

(b) What is the gradient descent update step for logistic regression with the cross entropy loss? (1 point)

2.2 Hate Speech Problem

Let us now tackle the problem of detecting hate speech in text. The United Nations defines hate speech as:

"any kind of communication in speech, writing or behaviour, that attacks or uses pejorative or discriminatory language with reference to a person or a group on the basis of who they are, in other words, based on their religion, ethnicity, nationality, race, colour, descent, gender or other identity factor."

To detect hate speech, let's use the following features:

Var	Definition
x_1	count(positive words)
x_2	count(negative words)
x_3	$\begin{cases} 1, & \text{if "hate"} \in \text{doc} \\ 0, & \text{otherwise} \end{cases}$
x_4	$\log_e$(word count of sample)

We have computed these features for a few documents giving us these values:

Doc	x_1	x_2	x_3	x_4	y
1	5	2	0	ln(43)	0
2	3	1	0	ln(20)	0
3	6	0	0	ln(78)	0
4	1	4	1	ln(28)	1
5	2	3	1	ln(38)	1
6	2	7	0	ln(67)	1

(a) Let the weights be $w = [-0.25, 0.3, 0.1, 0.04]$ and $b = 0.2$. Calculate the probability of each of the documents being hate speech $p(hateSpeech|doc)$ and assign labels to them using the sigmoid function. If $p(hateSpeech|doc) >= 0.5$, then the document would be classified as hate speech. (3 points)

(b) Assume the learning rate $\eta = 0.1$ and perform one update to the weights using gradient descent. Remember to average the update by the number of data points used. (2 points)

(c) Calculate the probabilities and labels again with the updated weights. Was there any change? (3 points)

(d) What do the weight updates tell you about the features we chose? What other features do you think would be helpful for this task?(2 points)

3 Programming

In this problem, you will do text classifications for Hate Speech. You need to both answer the questions and submit your code.

As stated in the previous problem, hate speech is a

1. **deliberate attack,**
2. **directed towards a specific group of people,**
3. **motivated by aspects of the group's identity.**

The three premises must be true for a sentence to be categorized as HATE. Here are two examples:

1. "Poor white kids being forced to treat apes and parasites as their equals."
2. "Islam is a false religion however unlike some other false religions it is crude and appeals to crude people such as arabs."

In (a), the speaker uses "apes" and "parasites" to refer to children of dark skin and implies they are not equal to "white kids". That is, it is an attack to the group composed of children of dark skin based on an identifying characteristic, namely, their skin colour. Thus, all the premises are true and (a) is a valid example of HATE. Example (b) brands all people of Arab origin as crude. That is, it attacks the group composed of Arab people based on their origin. Thus, all the premises are true and (b) is a valid example of HATE.

This problem will require programming in **Python 3**. The goal is to build a **Naive Bayes model** and a **logistic regression model** that you learnt from the class on a real-world hate speech classification dataset. Finally, you will explore how to design better features and improve the accuracy of your models for this task.

The dataset you will be using is collected from Twitter online. Each example is labeled as 1 (hatespeech) or 0 (Non-hatespeech). To get started, you should first download the data and starter code from the following location:

https://www.cc.gatech.edu/classes/AY2021/cs4650_spring/programming/h1.zip.

Try to run:

```
python main.py --model AlwaysPredictZero
```

This will load the data and run a default classifier `AlwaysPredictZero` which always predicts label 0 (non-hatespeech). You should be able to see the reported train accuracy = 0.4997. That says, always predicting non-hatespeech isn't that good. Let's try to build better classifiers!

Note that you need to implement models without use of any machine learning packages such as `sklearn`. Your solution should use simple Python lists or `numpy` arrays. (`numpy` arrays are recommended since they are much faster than Python lists. Your program should finish running within an acceptable time limit in Gradescope.) Note the Gradescope machine will not have any external packages installed so make sure to also delete/comment any statements importing external packages. We will only provide a training set, and evaluate your code based on our test set.

To have a quick check with your implementations, you can randomly split the dataset we give you into train and test set at a ratio of 8:2, compare the accuracy between the models you have implemented and related models in `sklearn` packages. You are encouraged to try `sklearn` in non-submitted code, to get an idea of what the expected outputs should and how much training time your models should require. None of the models you implement for this question should take longer than a few minutes to train.

Once you are done, please **only submit** your `utils.py` and `classifiers.py` to Gradescope.

1. **(Naive Bayes)** In this part, you should implement a Naive Bayes model with add-1 smoothing, as taught in the class. You are required to implement the `NaiveBayesClassifier` class in `classifiers.py`. You would probably want to take a look at the `UnigramFeature` class in `utils.py` that we have implemented for you already. After you finish coding, run `python main.py --model NaiveBayes` to check

the performance.
We will test your implementation on our test set. [15 points if `accuracy` ≥ 0.65, 9 pts if `accuracy` ≥ 0.60]

List the 10 words that, under your model, have the highest ratio of $\frac{P(w|1)}{P(w|0)}$ (the most distinctly hate-speech words). List the 10 words with the lowest ratio. What trends do you see? [10 points]

2. **(Logistic Regression)** In this part, you will implement a Logistic Regression model. You are required to implement the `LogisticRegressionClassifier` class in `classifiers.py`. First, implement a logistic regression model without regularization and run `python main.py --model LogisticRegression`, compare the performance with your Naive Bayes approach. Describe how both of these models differ from the perceptron classifier we talked about in class. Next, we would like to experiment with L_2 regularization. Add L_2 regularization with different weights, such as $\lambda = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$.

   Describe what you observed. (Again, you may want to split the training set we give you into your own training and test sets to observe the performance.) [10 points]

   For the **code** submission, hard-code the best λ you observed. Your model accuracy on our test set must be greater than 0.68 to receive full credit. Similar to part (a), partial credit will be awarded for accuracy above 0.60. Please make sure your model doesn't train for too long (more than a few minutes) locally, otherwise gradescope will time out while grading your assignment and you will receive no credit. [15 points]

3. **(Bonus)** In the last part, you'll explore and implement a more sophisicated set of features. You need to implement the class `BigramFeature` or modify the class `CustomFeature` in `utils.py`. Here are some common strategies (you are welcome to implement some of them but try to come up with more!):

   (a) Remove stopwords (e.g. a, the, in),

   (b) Use a mixture of unigrams, bigrams or trigrams, (It may take exponentially more time to run bigram features than unigram features. Make sure total running time is within the acceptable time limit in Gradescope.)

   (c) Use TF-IDF (refer to http://www.tfidf.com/) features.

   `NaiveBayesClassifier` should be used as the classifier in this problem (we use it in Gradescope as well). Use your creativity for this problem and try to obtain an accuracy as high as possible on your test set. The `CustomFeature` class should contain any custom features you'd like to be used for this part. **Note**: if these two classes are unimplemented, Gradescope will not be able to run your code or show your score on leaderboard. Thus, you will not receive any bonus.

   Describe the features you implemented and your reasoning for this. [Bonus: 10 points]

   You will receive up to 10 bonus points: up to 5 points based on novel features you try, and the rest based on how well your model performs compared to other submissions:

$$\text{Bonus} = 5 + 5 \cdot \frac{1}{\text{rank}}$$

   e.g. if you rank first in the class, you will receive the full bonus points!